

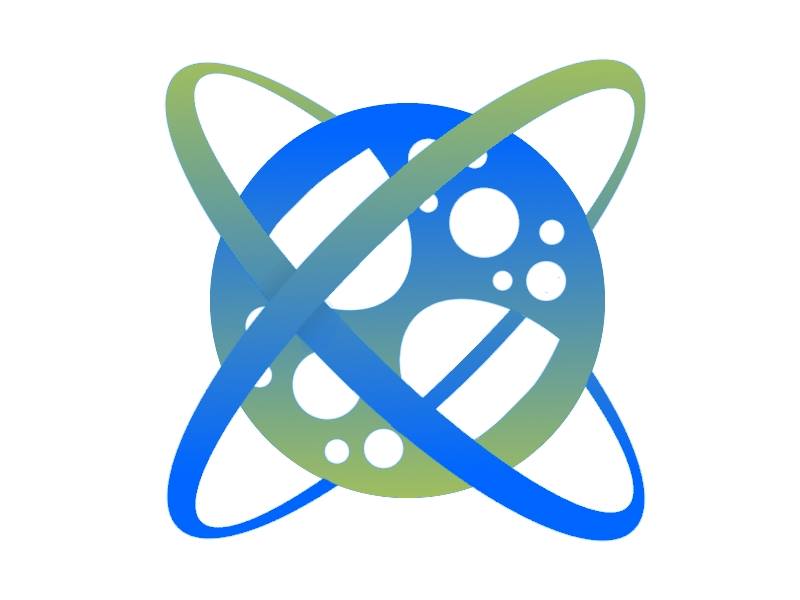
**Software Process, SOEN 341/4 S, Winter 2016**

Dr. Shang

Dr. Fancott

Mr. Morse

**TimeTurner** by team YAWD

****

**Project Design Document – Deliverable 2**

|  |  |
| --- | --- |
| **Team members information** | |
| **Name** | **SID** |
| **Daniel Di Corpo** | **26331602** |
| **Dimitri Topaloglou** | **29358269** |
| **Claudia Della Serra** | **26766048** |
| **Philip Lim** | **27485506** |
| **Aline Koftikian** | **27764162** |
| **Ryan Lee** | **27752504** |
| **Erin Benderoff** | **27768478** |
| **Ideawin-Bunthy Koun** | **26314155** |
| **Kevin Yasmine** | **27195346** |
| **Marc-Andre Leclair** | **27754876** |
| **Lori Dalkin** | **27738293** |
| **Bryce Drewery-Schoeler** | **27283199** |

Grading Sheet

|  |  |  |
| --- | --- | --- |
| Section | Evaluation criteria (see instructions in the template for details) | Grading |
| all | 10 marks are allocated for excellence, professionalism and quality of work above and beyond the correct meeting of specifications.. | /10 |
| 1 | Presentation of this document | /5 |
| 2 | Completeness and accuracy with regard to initial project description | /1 |
| 3.1 .  .  3.2  3.3 | Completeness and accuracy of the project functional requirements expressed as formal use cases, including difficulty and importance indicators  completeness and accuracy of the diagram and description of the domain model  completeness and accuracy with regard to initial project description accuracy with regard to initial project description, difficulty and importance ratings | /15  .  /3 .  /1 |
| 4.1 | Description of all team members’ capacities and schedule restrictions | /1 |
| 5 | List of goals removed from the project.  For each goal removed, give justifications in light of the resources available | /`1 |
| 6.1 .  6.2 | Clarity of textual description, validity of rationale, clarity and appropriateness of diagram, list of modules responsibilities  List of technologies used, validity of rationale | /2 .  /1 |
| 7.1 .  7.2 .  7.3 .  7.4  7.5  7.6 | Completeness of list of activities, clarity of their stated purpose, as well as statement of what artifacts they are producing  Completeness of list of artifacts to be produced during the project, validity of roles description of each artifact  Cost estimation of each individual artifact, validity of explanation of cost estimation, total cost estimate  Mapping of activities to individual project members  Accurate and complete presentation of milestones  *Assessment of risks `* | /1 .  /2 .  /2 .  /1  /1  /1 |
| 8 | Early Prototyping | /2 |
| Total |  | /50 |

DO NOT REMOVE THIS PAGE WHEN SUBMITTING YOUR DOCUMENT

**TABLE OF CONTENTS**

[1. Introduction 7](#_Toc446360806)

[2. Project Description 7](#_Toc446360807)

[3. Architectural Design 8](#_Toc446360808)

[3.1 Architecture Diagram (4+1 View) 8](#_Toc446360809)

[3.1.1 Physical View 8](#_Toc446360810)

[3.1.2 Logical View 9](#_Toc446360811)

[3.1.3 Process View 10](#_Toc446360813)

[3.1.4 Development View 10](#_Toc446360814)

[3.1.5 Situational View (Use Cases) 11](#_Toc446360815)

[3.2 Subsystem Interfaces Specifications 12](#_Toc446360816)

[3.2.1 IUserInteraction 12](#_Toc446360817)

[3.2.1.1 Specifications for methods implemented by the Student class 12](#_Toc446360818)

[3.2.1.2 Specifications for methods implemented by the Administrator class 15](#_Toc446360819)

[3.2.1.3 Specifications for methods implemented by the SavedUsers class 16](#_Toc446360820)

[3.2.2 IScheduleManagement 16](#_Toc446360821)

[3.2.2.1 Specifications for methods implemented by the Schedule class 17](#_Toc446360822)

[3.2.2.2 Specifications for methods implemented by the SavedSchedules class 19](#_Toc446360823)

[3.2.3 ICourseManagement 20](#_Toc446360824)

[3.2.3.1 Specifications for methods implemented by the Course class 20](#_Toc446360825)

[3.2.3.2 Specifications for methods implemented by the CoursesPassed class 22](#_Toc446360826)

[3.2.3.3 Specifications for methods implemented by the Section class 23](#_Toc446360827)

[3.2.3.4 Specifications for methods implemented by the Prerequisites class 24](#_Toc446360828)

[3.2.4 IPreferenceManagement 24](#_Toc446360829)

[3.2.4.1 Specifications for methods implemented by the Preferences class 25](#_Toc446360830)

[3.2.5 ISequenceManagement 25](#_Toc446360831)

[3.2.5.1 Specifications for methods implemented by the Sequence class 25](#_Toc446360832)

[4. Detailed Design 27](#_Toc446360833)

[4.1 Detailed Design Diagram and Unit Descriptions 27](#_Toc446360834)

[4.1.1 iUserInteraction 27](#_Toc446360835)

[4.1.2 IScheduleManagement 29](#_Toc446360836)

[4.1.3 ICourseManagement 31](#_Toc446360837)

[4.1.4 IPreferenceManagement 33](#_Toc446360838)

[4.1.5 ISequenceManagement 34](#_Toc446360839)

[5. Dynamic Design Scenarios 34](#_Toc446360840)

[5.1 Use Case 1: Login to TimeTurner 34](#_Toc446360841)

[5.1.1 Fully Dressed Use Case 34](#_Toc446360842)

[5.1.2 5.1.2 System Sequence Diagrams 36](#_Toc446360843)

[5.1.3 5.1.3 Contract Diagrams 37](#_Toc446360844)

[5.2 Use Case 2 : Logout of TimeTurner 39](#_Toc446360845)

[5.2.1 Fully Dressed Use Case 39](#_Toc446360846)

[5.2.2 System Sequence Diagrams 40](#_Toc446360847)

[5.2.3 Contract Diagrams 41](#_Toc446360848)

[5.3 Use Case 3: Create Course Sequence 43](#_Toc446360849)

[5.3.1 Fully Dressed Use Case 43](#_Toc446360850)

[5.3.2 System Sequence Diagram 44](#_Toc446360851)

[5.3.3 Contract Diagrams 44](#_Toc446360852)

[5.4 Use Case 4: Browse Course List 47](#_Toc446360853)

[5.4.1 Fully Dressed Use Case 47](#_Toc446360854)

[5.4.2 System Sequence Diagram 48](#_Toc446360855)

[5.4.3 Contract Diagrams 48](#_Toc446360856)

[5.5 Use Case 5: View Course Sequence 51](#_Toc446360857)

[5.5.1 Fully Dressed Use Case 51](#_Toc446360858)

[5.5.2 System Sequence Diagram 52](#_Toc446360859)

[5.5.3 Contract diagrams 52](#_Toc446360860)

[5.6 Use Case 6: Generate Schedule 54](#_Toc446360861)

[5.6.1 Fully Dressed Use Case 54](#_Toc446360862)

[5.6.2 5.6.2 System Sequence Diagram 55](#_Toc446360863)

[5.6.3 Contract Diagrams 55](#_Toc446360864)

[5.7 Use Case 7 – View Saved Schedules 57](#_Toc446360865)

[5.7.1 Fully Dressed Use Case 57](#_Toc446360866)

[5.7.2 System Sequence Diagram 58](#_Toc446360867)

[5.7.3 Operating Contract Diagrams 59](#_Toc446360868)

[5.8 Use Case 8 – View Academic Record 61](#_Toc446360869)

[5.8.1 Fully Dressed Use Case 61](#_Toc446360870)

[5.8.2 System Sequence Diagram 62](#_Toc446360871)

[5.8.3 Operating Contract Diagrams 62](#_Toc446360872)

[5.9 Use Case 9 – Drop Course 64](#_Toc446360873)

[5.9.1 Fully Dressed Use Case 64](#_Toc446360874)

[5.9.2 System Sequence Diagram 65](#_Toc446360875)

[5.9.3 Operating Contract Diagrams 65](#_Toc446360876)

[5.10 Use Case 10 – Add Course 69](#_Toc446360877)

[5.10.1 Fully Dressed Use Case 69](#_Toc446360878)

[5.10.2 System Sequence Diagram 70](#_Toc446360879)

[5.10.3 Operating Contract Diagrams 70](#_Toc446360880)

[5.11 Use Case 11 – Save Generated Schedule 74](#_Toc446360881)

[5.11.1 Fully Dressed Use Case 74](#_Toc446360882)

[5.11.2 Operating Contract Diagrams 75](#_Toc446360883)

[5.12 Use Case 12 – View Weekly Schedule 78](#_Toc446360884)

[5.12.1 Fully Dressed Use Case 78](#_Toc446360885)

[5.12.2 System Sequence Diagram 79](#_Toc446360886)

[5.12.3 Operating Contract Diagrams 79](#_Toc446360887)

[6. Estimation 81](#_Toc446360888)

[6.1 Function Point Estimation 81](#_Toc446360889)

[6.1.1 Unadjusted Function Points 81](#_Toc446360890)

[6.2 Module Estimations 82](#_Toc446360891)

[6.2.1 6.3.4 Total Deliverable Estimates 83](#_Toc446360892)

[6.3 Updated Gantt Chart 83](#_Toc446360893)

[7. Rapid Prototyping and Risk 85](#_Toc446360894)

[7.1 User Interface Mockup 85](#_Toc446360895)

[7.1.1 Main Layout 85](#_Toc446360896)

[7.1.2 Login Interface Mockup 86](#_Toc446360897)

[7.1.3 Generate Schedule Interface Mockup 87](#_Toc446360898)

[7.2 Risk 87](#_Toc446360899)

[7.2.1 Framework 87](#_Toc446360900)

[7.2.2 Time Constraint 88](#_Toc446360901)

[7.2.3 Control Version System 88](#_Toc446360902)

[7.2.4 Server Uptime 88](#_Toc446360903)

# Introduction

The following section covers the design of TimeTurner in detail. This design is partitioned into different views and diagrams, starting with the 4+1 architecture view. This provides 5 individual views on the system which include logical, development, process, physical and scenario views. Following, will be descriptions of the subsystems, along with the classes included in each. Accompanying each class will be an in depth description of the class and their respective class diagrams. The final section includes all the dynamic design scenarios for each given situation defined by the use cases.

# Project Description

The proposed and outlined web application, known as TimeTurner, is designed to auto-generate a student’s course sequence from their first semester up until the end of their degree. It takes into account user input preferences and and any previously completed courses or course prerequisites before creating this sequence. Preferences can be made by the student and include options such as night classes or having particular days off. The application will notify the user if a certain preference suggested results in an impossibility or conflict in the sequence. This sequence generator will be able to create a sequence at any point throughout the user’s degree, if sudden change in circumstances were to arise.

The goal of this application is to simplify the method with which students may decide and schedule their courses. If a course must be redone, the generator can decide what other courses should move where in regards to the remaining courses to be completed, which can be done in seconds, rather than hours. It saves the time of the user, in a simple and efficient manner. Ultimately, the system’s end goal will be to simplify a student’s task of creating their own course schedule in order to allow students to redirect their time to other more important activities, thus making course registration much simpler, quicker, and easier.

# Architectural Design

## Architecture Diagram (4+1 View)

Physical View

Logical View

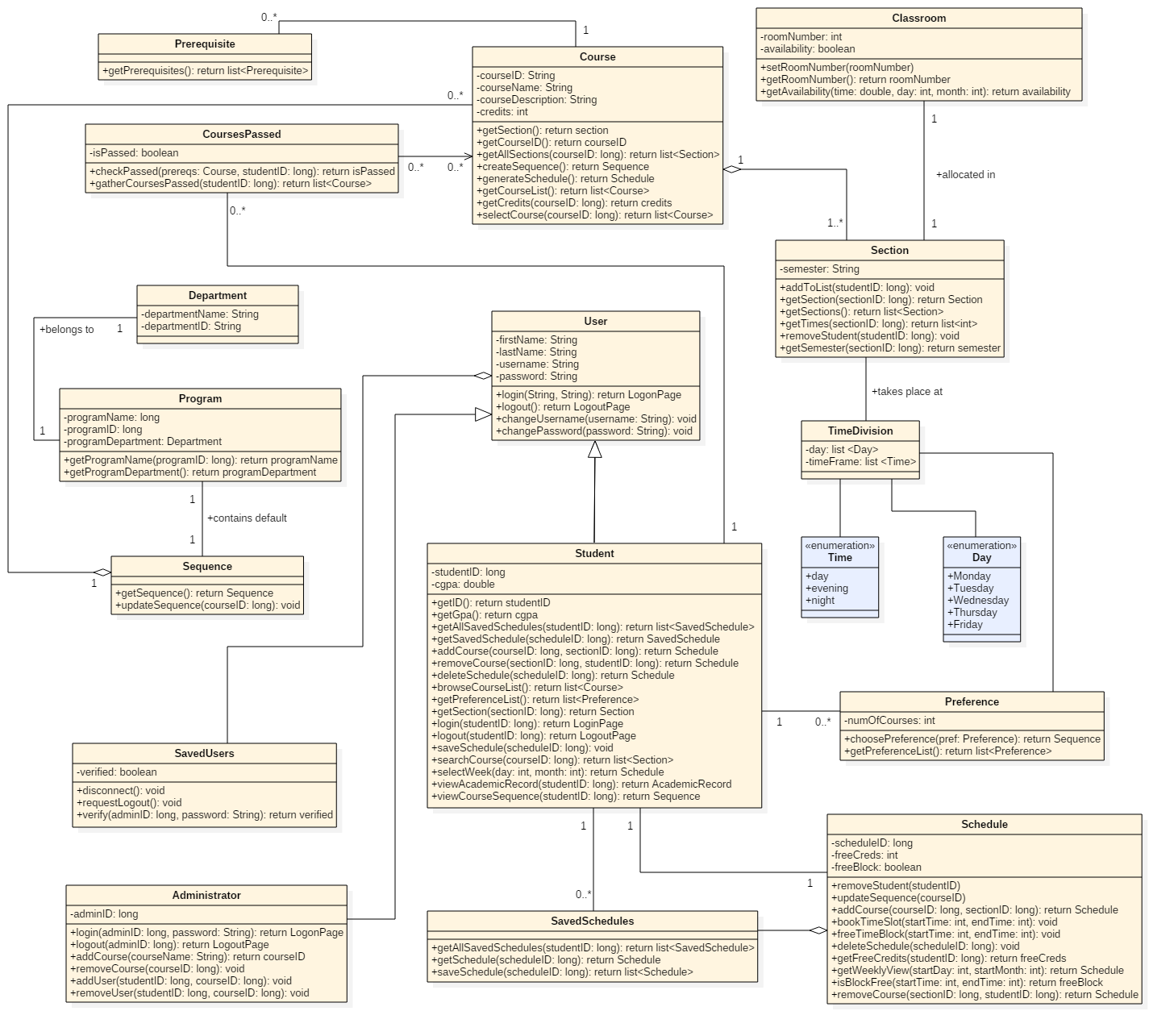


Figure 3.1.2 Class diagram representing the logical view

The logical view of the system’s architecture is represented by a full class diagram. The Student and Administrator classes are subclasses of the User superclass, where they inherit the methods of the latter, and overwrite the login() and logout() functions. The User class is an aggregate of SavedUsers, since the SavedUsers is comprised of many Users.

A student has a number of saved schedules, represented by the SavedSchedules class, a single schedule, and a number of preferences, which are the students preferred times and days of the week for classes.

The Section class is also related to the TimeDivision class by being offered at those times. Sections are also taken place in classrooms, therefore explaining the relationship between the two classes. It’s also related to the Course class through aggregation.

The Course class is an aggregation of the Section class, because a course is composed of many sections. A course has a number of prerequisites represented by the Prerequisites class, and a class CoursesPassed inherits from the Course class, which represents the courses from the Course class that have or have not been credited.

The Sequence class is an aggregation of the Course class, because a sequence is composed of courses. A program also contains its own default sequence of courses, which is shown with the relationship between the Program and Sequence classes. In addition, a program belongs to a department.

Process View

Development View

Situational View (Use Cases)

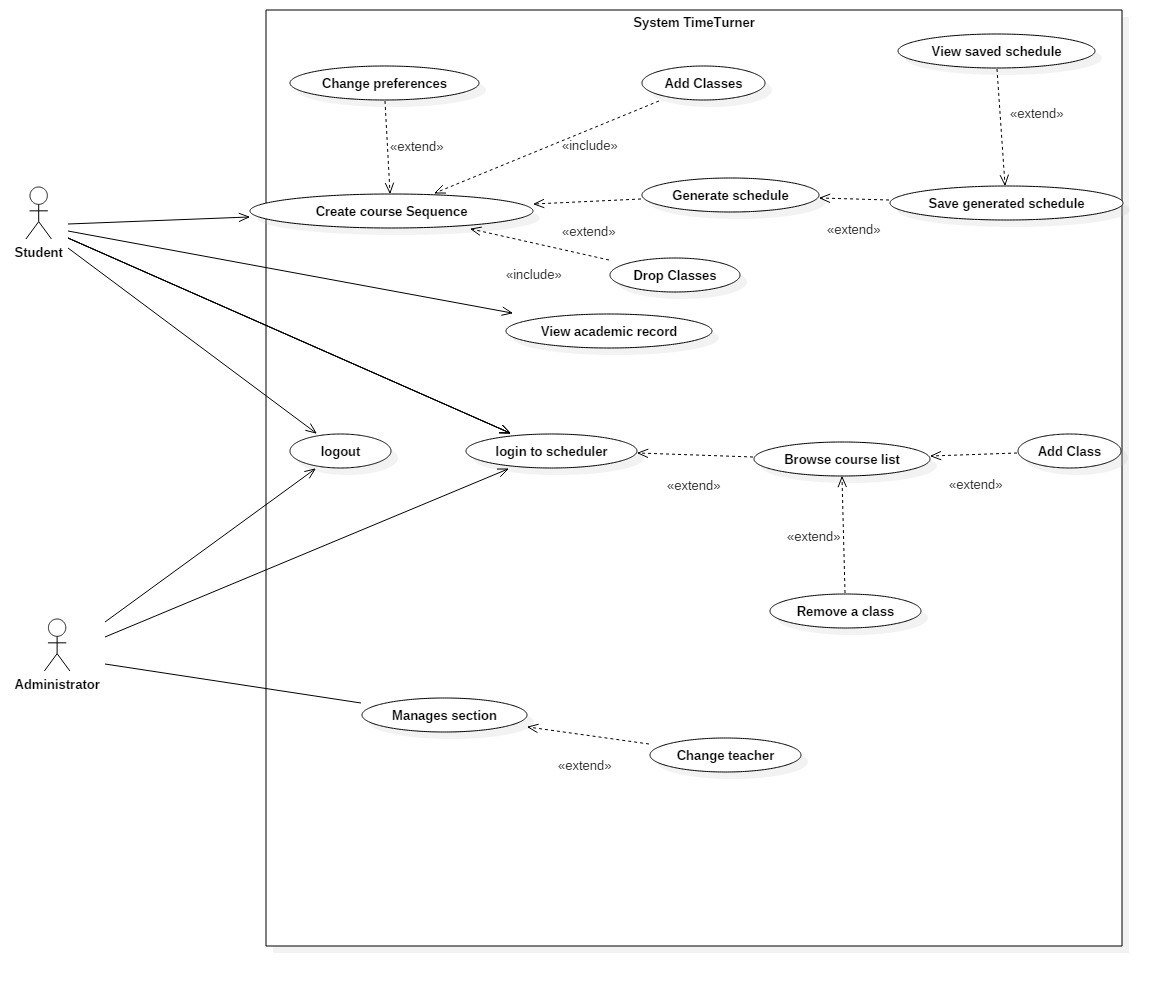


Figure 3.1.5 Use case diagram representing the situational view

The situational, or “+1”, view of the system’s architecture is represented by a full use case diagram. The use cases of the Student and Administator users are depicted in the diagram, represented by the ellipses.

The actions a student can perform are logging into the scheduler, creating their course sequence, viewing their academic record, and logging out of the scheduler. Creating the course sequence is comprised of several actions, including changing the preferences of the sequence, adding classes to the sequence, dropping classes, and generating schedule. The generating schedule use case is an extension of the saving schedule use case, which is an extension of viewing the schedule use case.

The administrator ‘s actions include login, logout, and manage sections. Similar to the student’s use cases, the login use case is an extension of the browsing course list use case, which is an extension of adding and removing classes. In addition, the administrator manages a section, which includes the action of changing the section’s teacher.

## Subsystem Interfaces Specifications

The system consists of five interfaces: IUserInteraction, IScheduleManagement, ICourseManagement, IPreferenceManagement and ISequenceManagement. Each of those interfaces provides a set of methods that components can publicly access. The classes that implements those interfaces are the ones presented in the architecture diagram, and the use of the method calls are explained in more detail in dynamic design scenario.

IUserInteraction

The user interaction interface provides basic methods that a user, either an administrator or a student, will use to interact with the system. Those interactions include login and logout operations. This interface also allows to consult data related to courses and schedules, but no modification on those data is allowed through this interface. The classes implementing the provided methods are the Student, Administrator and SavedUsers classes.

Specifications for methods implemented by the Student class

|  |
| --- |
| **Method** |
| browseCourseList() list of Course objects |
| **Specification** |
| ***Description:***  Retrieves and returns a list of courses that a student can take.  ***Input parameter(s):***  none  ***Return type:***  list of Course objects |

|  |
| --- |
| **Method** |
| getSavedSchedule(scheduleID:long) Schedule |
| **Specification** |
| ***Description:***  Given a schedule ID, retrieves and returns a specific saved schedule that represents the schedule of a semester.  ***Input parameter(s):***  scheduleID:long representing the schedule ID  ***Condition(s) for validity:***  scheduleID ≥ 0  ***Return type:***  Schedule |

|  |
| --- |
| **Method** |
| login(studentID:long, password:String) LogonPage |
| **Specification** |
| ***Description:***  Logs a student into the system, provided that the student ID and password passed as parameters are valid.  ***Input parameter(s):***  studentID:long representing the ID of the student to be logged in  password:String representing the password entered by the user  ***Conditions for validity:***  studentID ≥ 0  password is not empty  ***Return type:***  LogonPage |

|  |
| --- |
| **Method** |
| logout(studentID:long) LogoutPage |
| **Specification** |
| ***Description:***  Logs a student out of the system.  ***Input parameter(s):***  studentID:long representing the ID of the student to be logged out  ***Conditions for validity:***  studentID ≥ 0  ***Return type:***  LogoutPage |

|  |
| --- |
| **Method** |
| searchCourse(courseID:long) list of Section objects |
| **Specification** |
| ***Description:***  Retrieves and returns information about a course, including the course’s sections, given the course ID.  ***Input parameter(s):***  courseID:long representing the course ID  ***Conditions for validity:***  courseID ≥ 0  ***Return type:***  list of Section objects |

|  |
| --- |
| **Method** |
| selectWeek(day:int, month:int) Schedule |
| **Specification** |
| ***Description:***  Provides the caller with a weekly schedule view given the day and month as valid integer values for the targeted week.  ***Input parameter(s):***  day:int representing a calendar day in the targeted week  month:int representing the month of the targeted week  ***Conditions for validity:***  1 ≤ day ≤ 31  1 ≤ month ≤ 12  ***Return type:***  Schedule |

|  |
| --- |
| **Method** |
| viewAcademicRecord(studentID:long) AcademicRecord |
| **Specification** |
| ***Description:***  Allows to view the academic record of a student given the student’s ID.  ***Input parameter(s):***  studentID:long representing the ID of the student  ***Conditions for validity:***  studentID ≥ 0  ***Return type:***  AcademicRecord |

|  |
| --- |
| **Method** |
| viewCourseSequence(studentID:long) Sequence |
| **Specification** |
| ***Description:***  Retrieves and returns the sequence of a student given the ID of the student.  ***Input parameter(s):***  studentID:long representing the ID of the student  ***Conditions for validity:***  studentID ≥ 0  ***Return type:***  Sequence |

Specifications for methods implemented by the Administrator class

|  |
| --- |
| **Method** |
| login(adminID:long, password:String) LogonPage |
| **Specification** |
| ***Description:***  Logs an administrator into the system, provided that the admin ID and password passed as parameters are valid.  ***Input parameter(s):***  adminID:long representing the ID of the administrator to be logged in  password:String representing the password entered by the user  ***Conditions for validity:***  adminID ≥ 0  password is not empty  ***Return type:***  LogonPage |

|  |
| --- |
| **Method** |
| logout(adminID:long) LogoutPage |
| **Specification** |
| ***Description:***  Logs an administrator out of the system.  ***Input parameter(s):***  adminID:long representing the ID of the administrator to be logged out  ***Conditions for validity:***  adminID ≥ 0  ***Return type:***  LogoutPage |

Specifications for methods implemented by the SavedUsers class

|  |
| --- |
| **Method** |
| disconnect() |
| **Specification** |
| ***Description:***  Disconnects a user from the system.  ***Input parameter(s):***  none  ***Return type:***  void |

|  |
| --- |
| **Method** |
| requestLogout() |
| **Specification** |
| ***Description:***  Logs a user out of the system.  ***Input parameter(s):***  none  ***Return type:***  void |

|  |
| --- |
| **Method** |
| verify(userID:long, password:String) boolean |
| **Specification** |
| ***Description:***  Verifies that a userID matches the password associated with it. Returns true if the given information is valid, false otherwise.  ***Input parameter(s):***  userID:long representing the ID of the user  password:String representing the password the user entered  ***Conditions for validity:***  userID ≥ 0  password is not empty  ***Return type:***  boolean |

IScheduleManagement

The IScheduleManagement interface consists of a set of methods that allow another component to access and modify data about a schedule. The classes Schedule and SavedSchedules implement the provided methods.

Specifications for methods implemented by the Schedule class

|  |
| --- |
| **Method** |
| addCourse(courseID:long, sectionID:long) Schedule |
| **Specification** |
| ***Description:***  Adds a course given the course ID and the section ID to a student’s schedule.  ***Input parameter(s):***  courseID:long representing the ID of the course  sectionID:long representing the ID of the section  ***Conditions for validity:***  courseID ≥ 0  sectionID ≥ 0  ***Return type:***  Schedule |

|  |
| --- |
| **Method** |
| bookTimeSlot(startTime:int, endTime:int) |
| **Specification** |
| ***Description:***  Sets time slots in a scheduled as occupied given the starting and ending time of the time block to book.  ***Input parameter(s):***  startTime:int representing the time at which the time block starts  endTime:int representing the time at which the time block ends  ***Conditions for validity:***  0 ≤ startTime ≤ endTime  ***Return type:***  void |

|  |
| --- |
| **Method** |
| deleteSchedule(scheduleID:long) |
| **Specification** |
| ***Description:***  Deletes a schedule given its schedule ID.  ***Input parameter(s):***  scheduleID:long representing the ID of the schedule to be deleted  ***Conditions for validity:***  scheduleID ≥ 0  ***Return type:***  void |

|  |
| --- |
| **Method** |
| freeTimeBlock(startTime:int, endTime:int) |
| **Specification** |
| ***Description:***  Frees a time block of a schedule given the start and end time of the block.  ***Input parameter(s):***  startTime:int representing the time at which the time block starts  endTime:int representing the time at which the time block ends  ***Conditions for validity:***  0 ≤ startTime ≤ endTime  ***Return type:***  void |

|  |
| --- |
| **Method** |
| getFreeCredits(studentID:long) int |
| **Specification** |
| ***Description:***  Returns the number of free credits a student is still allowed to take.  ***Input parameter(s):***  studentID:long representing the ID of the student  ***Conditions for validity:***  studentID ≥ 0  ***Return type:***  int |

|  |
| --- |
| **Method** |
| getWeeklyView(startDay:int, startMonth:int) Schedule |
| **Specification** |
| ***Description:***  Generates a weekly view of a schedule given the starting date.  ***Input parameter(s):***  startDay:int representing the day on which the week starts  startMonth:int representing the month in which the week is found  ***Conditions for validity:***  1 ≤ startDay ≤ 31  1 ≤ startMonth ≤ 12  ***Return type:***  Schedule |

|  |
| --- |
| **Method** |
| isBlockFree(startTime:int, endTime:int) boolean |
| **Specification** |
| ***Description:***  Verifies whether a time block defined by the given starting and ending time is free in a schedule.  ***Input parameter(s):***  startTime:int representing the time at which the time block starts  endTime:int representing the time at which the time block ends  ***Conditions for validity:***  0 ≤ startTime ≤ endTime  ***Return type:***  boolean |

|  |
| --- |
| **Method** |
| removeCourse(sectionID:long, studentID:long) Schedule |
| **Specification** |
| ***Description:***  Removes a given course section from the schedule of a student.  ***Input parameter(s):***  sectionID:long representing the ID of the section to be removed  studentID:long representing the ID of the student  ***Conditions for validity:***  sectionID ≥ 0  studentID ≥ 0  ***Return type:***  Schedule |

Specifications for methods implemented by the SavedSchedules class

|  |
| --- |
| **Method** |
| getAllSavedSchedules(studentID:long) list of Schedule objects |
| **Specification** |
| ***Description:***  Retrieves and returns all saved schedules of a student given its student ID.  ***Input parameter(s):***  studentID:long representing the student ID  ***Condition(s) for validity:***  studentID ≥ 0  ***Return type:***  list of Schedule objects |

|  |
| --- |
| **Method** |
| getSchedule(scheduleID:long) Schedule |
| **Specification** |
| ***Description:***  Retrieves and returns the schedule of a semester given the ID of the schedule.  ***Input parameter(s):***  scheduleID:long representing the ID of the schedule  ***Conditions for validity:***  scheduleID ≥ 0  ***Return type:***  Schedule |

|  |
| --- |
| **Method** |
| saveSchedule(scheduleID:long) list of Schedule objects |
| **Specification** |
| ***Description:***  Saves a new schedule related to a student given the schedule ID, then returns an updated list of schedules that have been saved.  ***Input parameter(s):***  scheduleID:long representing the ID of the schedule to be saved  ***Conditions for validity:***  scheduleID ≥ 0  ***Return type:***  list of Schedule objects |

ICourseManagement

The ICourseManagement interface allows to consult information about academic courses as well as the sections of each course. It also allows to perform modification on courses, such as adding and removing students from a course. The ICourseManagement interface is also responsible for creating course sequences and generating schedules. The Course, CoursesPassed, Section and Prerequisites classes are the ones implementing the provided methods.

Specifications for methods implemented by the Course class

|  |
| --- |
| **Method** |
| createSequence() Sequence |
| **Specification** |
| ***Description:***  Creates and returns a course sequence for a student.  ***Input parameter(s):***  none  ***Return type:***  Sequence |

|  |
| --- |
| **Method** |
| generateSchedule() Schedule |
| **Specification** |
| ***Description:***  Generates and returns a student’s schedule.  ***Input parameter(s):***  none  ***Return type:***  Schedule |

|  |
| --- |
| **Method** |
| getAllSections(courseID:long) list of Section objects |
| **Specification** |
| ***Description:***  Returns a list containing all the sections of a course.  ***Input parameter(s):***  courseID:long representing the ID of the course to be consulted  ***Conditions for validity:***  courseID ≥ 0  ***Return type:***  list of Section objects |

|  |
| --- |
| **Method** |
| getCourseList() list of Course objects |
| **Specification** |
| ***Description:***  Returns a list containing all courses that a student can take.  ***Input parameter(s):***  none  ***Return type:***  list of Course objects |

|  |
| --- |
| **Method** |
| getCredits(courseID:long) int |
| **Specification** |
| ***Description:***  Returns the number of assigned credits of a course.  ***Input parameter(s):***  courseID:long representing the ID of the course  ***Conditions for validity:***  courseID ≥ 0  ***Return type:***  int |

|  |
| --- |
| **Method** |
| selectCourse(courseID:long) list of Section objects and Prerequisite objects |
| **Specification** |
| ***Description:***  Given the ID of the selected course, returns information including the sections and prerequisites of that course.  ***Input parameter(s):***  courseID:long representing the ID of the course  ***Conditions for validity:***  courseID ≥ 0  ***Return type:***  (mixed) list of Section objects and Prerequisite objects |

Specifications for methods implemented by the CoursesPassed class

|  |
| --- |
| **Method** |
| checkPassed(prereqs:Course, studentID:long) boolean |
| **Specification** |
| ***Description:***  Verifies that a student has passed a prerequisite course. Returns true if the student has passed, false otherwise.  ***Input parameter(s):***  courseID:long representing the ID of the prerequisite course  studentID:long representing the ID of the student to verify  ***Conditions for validity:***  courseID ≥ 0  studentID ≥ 0  ***Return type:***  Boolean |

|  |
| --- |
| **Method** |
| gatherCoursesPassed(studentID:long) list of Course objects |
| **Specification** |
| ***Description:***  Retrieves and returns a list of courses that a student has successfully completed.  ***Input parameter(s):***  studentID:long representing the ID of the student  ***Conditions for validity:***  studentID ≥ 0  ***Return type:***  list of Course objects |

Specifications for methods implemented by the Section class

|  |
| --- |
| **Method** |
| addToList(studentID:long) |
| **Specification** |
| ***Description:***  Adds a student to the list of registered student for a section given the student ID.  ***Input parameter(s):***  studentID:long representing the ID of the student  ***Conditions for validity:***  studentID ≥ 0  ***Return type:***  void |

|  |
| --- |
| **Method** |
| getSection(sectionID:long) Section |
| **Specification** |
| ***Description:***  Retrieves and returns information about a course given the section ID.  ***Input parameter(s):***  sectionID:long representing the ID of the section  ***Conditions for validity:***  sectionID ≥ 0  ***Return type:***  Section |

|  |
| --- |
| **Method** |
| getSections() list of Section objects |
| **Specification** |
| ***Description:***  Retrieves and returns a list of section associated with a course.  ***Input parameter(s):***  none  ***Return type:***  list of Section objects |

|  |
| --- |
| **Method** |
| getTimes(sectionID:long) list of int |
| **Specification** |
| ***Description:***  Returns a list of integers representing the starting and ending time of each time block that a section holds.  ***Input parameter(s):***  sectionID:long representing the ID of the section  ***Conditions for validity:***  sectionID ≥ 0  ***Return type:***  list of int |

|  |
| --- |
| **Method** |
| removeStudent(studentID:long) |
| **Specification** |
| ***Description:***  Removes a student from the list of registered student of a course.  ***Input parameter(s):***  studentID:long representing the ID of the student to be removed  ***Conditions for validity:***  studentID ≥ 0  ***Return type:***  void |

Specifications for methods implemented by the Prerequisites class

|  |
| --- |
| **Method** |
| getPrerequisites() list of Prerequisite objects |
| **Specification** |
| ***Description:***  Returns a list of prerequisites for a course, or an empty list if no prerequisite courses are required for a course.  ***Input parameter(s):***  none  ***Return type:***  list of Prerequisite objects |

IPreferenceManagement

The preference management interface allows to access and modify data related to the scheduling preferences of a student. The only class implementing the methods provided by this interface is the Preferences class.

Specifications for methods implemented by the Preferences class

|  |
| --- |
| **Method** |
| choosePreference(pref:Preference) Sequence |
| **Specification** |
| ***Description:***  Adds a preference to the list of scheduling preferences of a student and returns the list of all chosen preferences.  ***Input parameter(s):***  pref:Preference representing a chosen preference  ***Conditions for validity:***  pref != null  ***Return type:***  list of Preference objects |

|  |
| --- |
| **Method** |
| getPreferenceList() list of Preference objects |
| **Specification** |
| ***Description:***  Retrieves and returns a list of preferences assigned to a student.  ***Input parameter(s):***  none  ***Return type:***  list of Preference objects |

ISequenceManagement

The sequence management interface provides methods to access and modify the course sequence of a student. The only class implementing the provided methods is the Sequence class.

Specifications for methods implemented by the Sequence class

|  |
| --- |
| **Method** |
| getSequence() Sequence |
| **Specification** |
| ***Description:***  Returns an object representing a course sequence of a student.  ***Input parameter(s):***  none  ***Return type:***  Sequence |

|  |
| --- |
| **Method** |
| updateSequence(courseID:long) |
| **Specification** |
| ***Description:***  Updates a course sequence after a course removal given the ID of the course removed.  ***Input parameter(s):***  courseID:long representing the ID of the course removed  ***Conditions for validity:***  courseID ≥ 0  ***Return type:***  void |

# Detailed Design

## Detailed Design Diagram and Unit Descriptions

iUserInteraction

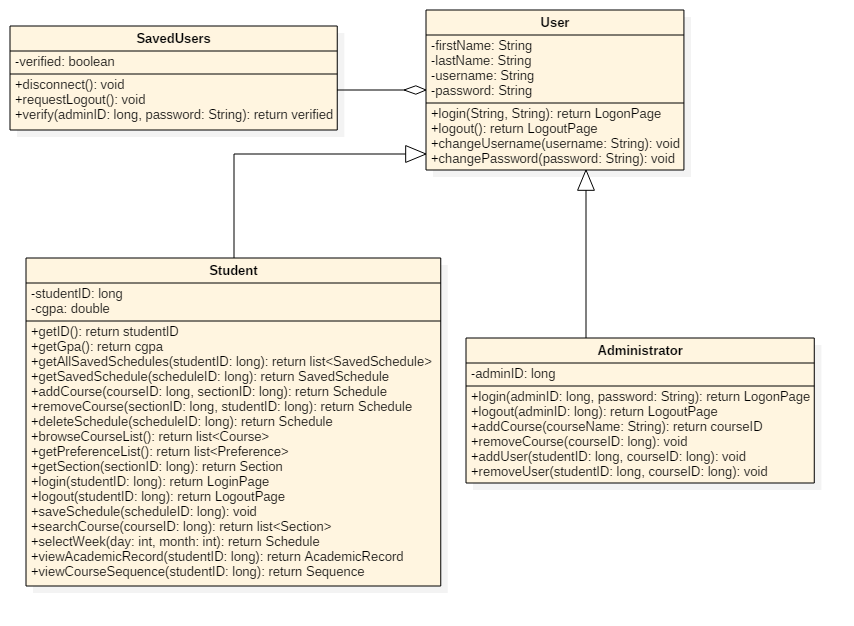


Figure 4.1.1 IUserInteraction subsystem class diagram

This subsystem is responsible to every interaction made between the User and the System, such as logging in, browsing the course list, setting preferences, adding courses, logging out, etc. The classes in this subsystem are the Student, Administrator and SavedUsers classes. Therefore, this subsystem represents the functions behind the user interface of the system.

|  |
| --- |
| **Student Class** |
| **Description** |
| This class models students, which are users that interact with the program directly. |
| **Attributes** |
| studentID: long  the studentID is a unique identifier for every student  cgpa: double  the cgpa is the cumulative GPA of each student |
| **Methods** |
| * getID() studentID   returns the student’s ID   * getGPA() cgpa   returns the student’s GPA   * getAllSavedSchedules(studentID:long) list of Schedule objects   returns a list of all the previous and upcoming schedules of a student   * getSavedSchedule(scheduleID:long) Schedule   returns the saved schedule of a student   * addCourse(courseID:long, sectionID:long) Schedule   adds a course with section sectionID to the student’s sequence   * removeCourse(sectionID:long, studentID:long) Schedule   removes a course from the student’s sequence   * deleteSchedule(scheduleID: long) Schedule   deletes the entire schedule   * browseCourseList() list of Course objects   returns a list of all the available courses   * getPreferenceList() list of Preference objects   returns the student’s preferences   * getSection(sectionID:long) Section   returns the section   * login(studentID:long, password:String) LogonPage   gives the student access to the system   * logout(studentID:long) LogoutPage   disconnects student from the system   * saveSchedule(scheduleID:long)   saves changes to the schedule   * searchCourse(courseID:long) list of Section objects   takes a course ID as input and returns the matched results   * selectWeek(day:int, month:int) Schedule   returns a schedule of the selecteddate   * viewAcademicRecord(studentID:long) AcademicRecord   returns the student’s academic record   * viewCourseSequence(studentID:long) Sequence   returns the student’s sequence |

|  |
| --- |
| **Administrator Class** |
| **Description** |
| Administrators are users that interact with the program directly. They log in, make changes in the schedule, then log out. |
| **Attributes** |
| adminID: long  the adminID is a unique identifier for each administrator |
| **Methods** |
| * login(adminID:long, password:String) LogonPage * logout(adminID:long) LogoutPage |

|  |
| --- |
| **SavedUsers Class** |
| **Description** |
| SavedUsers is the class that verifies if a user is legitimate or not, and allows it to logout. |
| **Attributes** |
| verified: Boolean  this attribute is the state of the user which determines whether his login is successful or not |
| **Methods** |
| * disconnect() void   disconnects the student from the system   * requestLogout() void   makes a request to disconnect   * verify(adminID: long, password: String) verified   verifies if the login info is correct |

IScheduleManagement

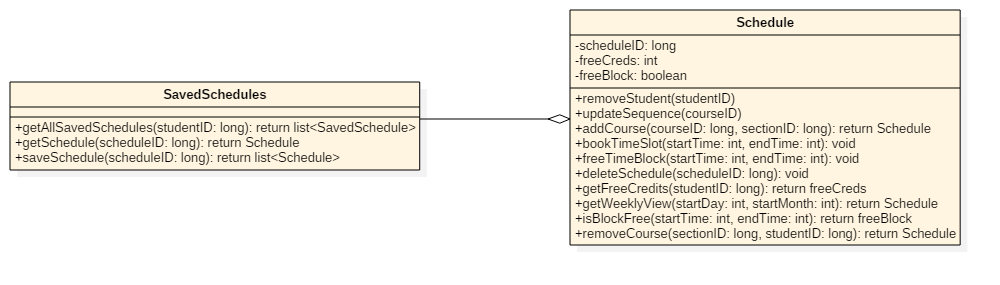


Figure 4.1.2 IScheduleManagement subsystem class diagram

The Schedule Management subsystem, consisting of the Schedule and SavedSchedules classes, is responsible for all schedule operations. In other words, whenever another one of the subsystems wants to modify or interact with a schedule or the list of schedules, they interact with this subsystem which in turn does the operations.

|  |
| --- |
| **Schedule Class** |
| **Description** |
| This class manages courses in a schedule, and ensures that courses don’t overlap |
| **Attributes** |
| scheduleID: long  every schedule has a unique long identifier  freeCreds: int  freeCreds is the amount of remaining free credits that the student can take in a semester  freeBlock: boolean  freeBlock is a Boolean which determines whether the time for a class is free or not |
| **Methods** |
| * removeStudent(studentID)   removes the student   * updateSequence(courseID)   updates the sequence after changes are made   * addCourse(courseID:long, sectionID:long) Schedule   adds a course to the schedule   * bookTimeSlot(startTime:int, endTime:int)   books a timeslot for a course to prevent overlap   * freeTimeBlock(startTime:int, endTime:int)   frees a timeslot that was booked   * deleteSchedule(scheduleID:long)   deletes a schedule   * getFreeCredits(studentID:long) int   returns the amount of credits still available   * getWeeklyView(startDay:int, startMonth:int) Schedule   returns a weekly view of the schedule   * isBlockFree(startTime:int, endTime:int) Boolean   computes if the timeslot is available and returns true or false   * removeCourse(sectionID:long, studentID:long) Schedule   removes a course from the schedule |

|  |
| --- |
| **SavedSchedules** |
| **Description** |
| This class manages schedules as a whole. It could get a schedule, save it, or get a list off all schedules that are in the system. |
| **Attributes** |
| None |
| **Methods** |
| * getAllSavedSchedules(studentID:long) list of Schedule objects   returns all schedules available in the system   * getSchedule(scheduleID:long) Schedule   returns a single schedule that has the id scheduleID   * saveSchedule(scheduleID:long) list of Schedule objects   saves changes made to schedules |

ICourseManagement

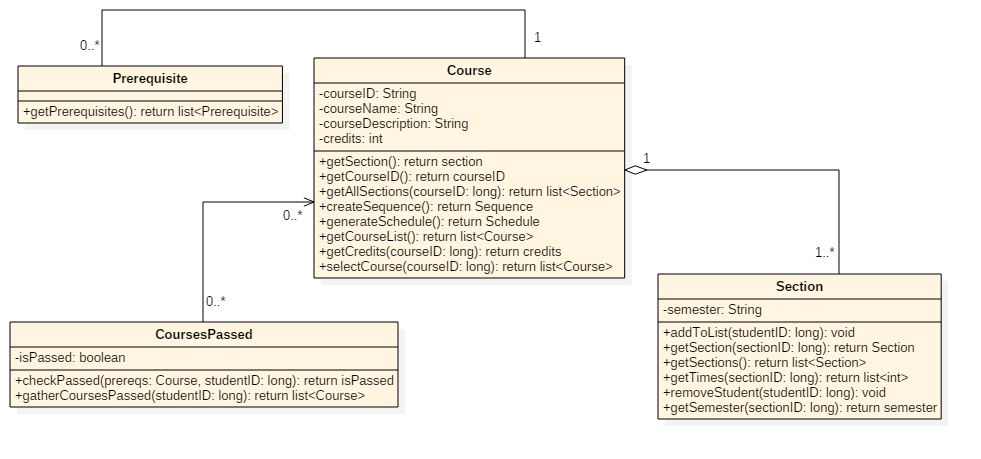


Figure 4.1.3 ICourseManagement subsystem class diagram

The Course, CoursesPassed, Section and Prerequisite classes make up this subsystem. Because the Courses are much more complex than a name (they contain sections, times, prerequisites, etc.), we cannot have a single class take care of everything. Instead, we have the Course Management subsystem which uses the above classes to modify and interact with the Courses.

|  |
| --- |
| **Course Class** |
| **Description** |
| The Course class contains all the information on the classes available. |
| **Attributes** |
| courseID: String  every course has a unique String identifier  courseName: String  every course has a name  courseDescription: String  courses have a String description  credits: int  this attribute is an integer value of the credits the course is worth |
| **Methods** |
| * getSection() section   returns a section of a class   * getCourseList() list of Course objects   returns a list of available courses   * getAllSections(courseID:long) list of Section objects   returns a list of all available sections   * createSequence() Sequence   initializes a sequence   * generateSchedule() Schedule   generates a schedule based on preferences   * getCourseList() list<Course>   gets a list of courses in a schedule   * getCredits(courseID:long) int   returns the credits of a course   * selectCourse(courseID:long) list of Section objects and Prerequisite objects   selects a course that has the ID courseID |

|  |
| --- |
| **CoursesPassed Class** |
| **Description** |
| This class is responsible for confirming and returning the passed courses. |
| **Attributes** |
| isPassed: Boolean  this attribute is a Boolean value that determines if a course is already passed or not |
| **Methods** |
| * checkPassed(prereqs:Course, studentID:long) boolean   checks if a course is passed by student with ID studentID   * gatherCoursesPassed(studentID:long) list of Course objects   returns a list of passed courses for student with ID studentID |

|  |
| --- |
| **Section Class** |
| **Description** |
| A course can have multiple sections; this class contains the different sections and is responsible for adding and removing students from the sections. |
| **Attributes** |
| semester: String  the semester String is a value which determines which semester the course is in |
| **Methods** |
| * addToList(studentID:long)   adds a student to a section   * getSection(sectionID:long) Section   returns a section with the ID sectionID   * getSections() list of Section objects   returns a list of available sections   * getTimes(sectionID:long) list of int   returns the time of section with ID sectionID   * removeStudent(studentID:long)   removes student with ID studentID from a section |

|  |
| --- |
| **Prerequisite Class** |
| **Description** |
| Each course has a list of prerequisites. This class returns this list. |
| **Attributes** |
| None |
| **Methods** |
| * getPrerequisites() list of Prerequisite objects   returns the prerequisites of a course |

IPreferenceManagement

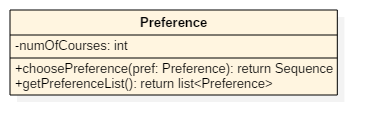


Figure 4.1.4 IPreferenceManagement subsystem class diagram

Though this subsystem only contains one class: the Preference class, tt is responsible for taking the students preferences and giving it to another subsystem to help create a schedule according to the student’s needs. These preferences dictate the sections that are shown to the student in order to choose from to build their schedules.

|  |
| --- |
| **Preference Class** |
| **Description** |
| This class manages the user’s preference and generates a sequence, and is capable of returning all the preferences in the system. |
| **Attributes** |
| numOfCourses: int  numOfCourses is an integer attribute that determines how many courses a student wishes to take |
| **Methods** |
| * choosePreference(pref:Preference) Sequence   applies the preferences set by the user and returns a sequence   * getPreferenceList() list of Preference objects   gets the preferences set by the user |

ISequenceManagement

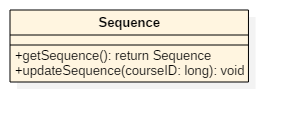


Figure 4.1.5 ISequenceManagement subsystem class diagram

This subsystem also contains one class, the Sequence class. It is responsible for returning the Sequence of a student to a user who wishes to view it, or to update the sequence by removing courses (when a student completes a course). The sequence dictates which classes a student may take in any given semester, and is updated when a student adds or drops a class on their schedule.

|  |
| --- |
| **Sequence Class** |
| **Description** |
| This class allows other classes to get the Sequence of a student, and the class also is capable of updating the sequence whenever a course is completed. |
| **Attributes** |
| None |
| **Methods** |
| * getSequence() Sequence   returns a sequence of a student   * updateSequence(courseID:long)   whenever a course is passed, this method is used to remove the course from the sequence |

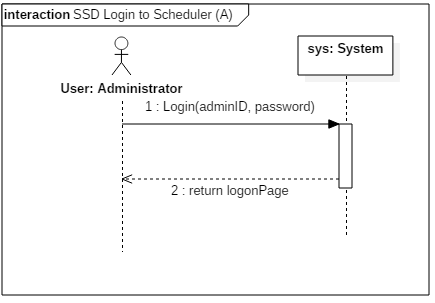
# Dynamic Design Scenarios

Use Case 1: Login to TimeTurner

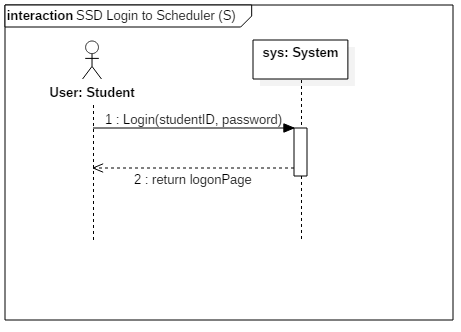
Fully Dressed Use Case

|  |  |  |
| --- | --- | --- |
| Use Case ID: | UC1 | |
| Use Case Name: | Login to TimeTurner | |
| Created By: | Marc-Andre Leclair | Last Updated By: Claudia Della Serra |
| Date Created: | January 25th, 2016 | Last Revision Date: February 8, 2016 |
| Actor(s): | Student, Administrator | |
| Goal/Actor Goals: | A student or Administrator wants to Login in their account | |
| Description/Summary: | The Student or Administrator wants to login into their account. The initial webpage contains a login box where the he or she can enter their username and password. The request is sent and it is check in the database whether or not the account exist or if the password is correct | |
| Preconditions: | * The user has an internet connection. * The user has valid login credentials. | |
| Post-conditions: | The user is authenticated and logged in. | |
| Minimum Guarantee: | User will not log in | |
| Basic Flow: | 1. Student enters its login information 2. Server verifies if it is in the database 3. Student is logged in and brought to the home page. | |
| Risk assessment: | Low | |
| Importance assessment: | 5/5 | |

5.1.2 System Sequence Diagrams



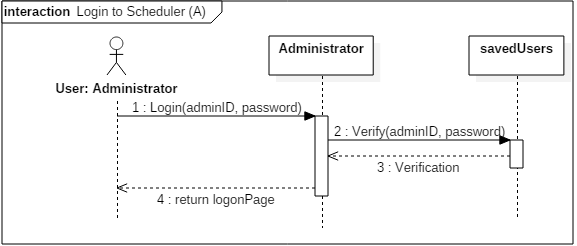
*Figure 5.1.1 Administrator login system sequence diagram*



*Figure 5.1.2 Student login system sequence diagram*

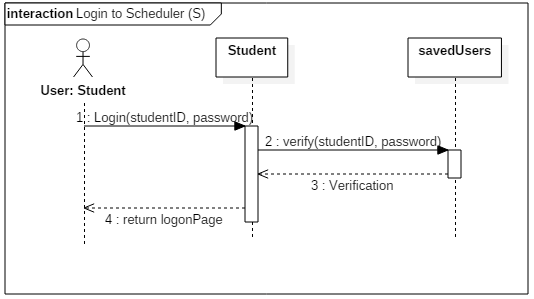
5.1.3 Contract Diagrams

|  |  |  |
| --- | --- | --- |
| Contract 1.1 | Login | |
| Created By: | Ryan Lee | Last Updated By: Ryan Lee |
| Date Created: | March 17, 2016 | Last Revision Date: March 17, 2016 |
| Operation: | login(adminID: long) | |
| Cross-reference: | UC1 | |
| Preconditions: | * The user has an internet connection. * The user has valid login credentials. | |
| Post-conditions: | * The user is logged into their account. (instance creation) * A connection to the servers is established. (association creation) | |



*Figure 5.1.3 Administrator login contract diagram*

|  |  |  |
| --- | --- | --- |
| Contract 1.2 | Login | |
| Created By: | Ryan Lee | Last Updated By: Ryan Lee |
| Date Created: | March 17, 2016 | Last Revision Date: March 17, 2016 |
| Operation: | login(studentID: long) | |
| Cross-reference: | UC1 | |
| Preconditions: | * The user has an internet connection. * The user has valid login credentials. | |
| Post-conditions: | * The user is logged in. (instance creation) * A connection to the servers is established. (association creation) | |



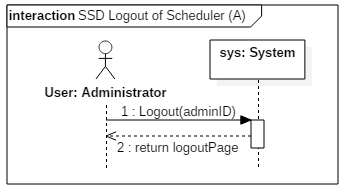
*Figure 5.1.4 Student login contract diagram*

Use Case 2 : Logout of TimeTurner

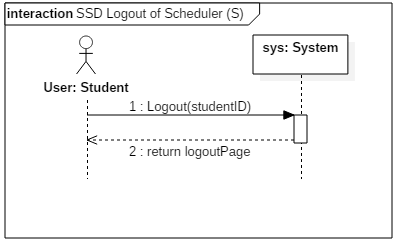
Fully Dressed Use Case

|  |  |  |
| --- | --- | --- |
| Use Case ID: | UC2 | |
| Use Case Name: | Logout of Scheduler | |
| Created By: | Marc-Andre Leclair | Last Updated By: Ryan Lee |
| Date Created: | January 25th, 2016 | Last Revision Date: March 17, 2016 |
| Actor(s): | Student, Administrator | |
| Goal/Actor Goals: | A student or Administrator wants to Logout of their account. | |
| Description/Summary: | The Student or Administrator wants to logout of their account. He or she sends a request to the server to close the connection between themselves and the Time Turner. | |
| Preconditions: | The user is logged in their account | |
| Post-conditions: | The user is disconnected from the server and logged out. | |
| Minimum Guarantee: | User will remain logged in | |
| Basic Flow: | 1. The user indicates that they wish to logout of the system. 2. The server terminates the connection. 3. The user is logged out through the webpage. | |
| Risk assessment: | Low | |
| Importance assessment: | 5/5 | |

System Sequence Diagrams



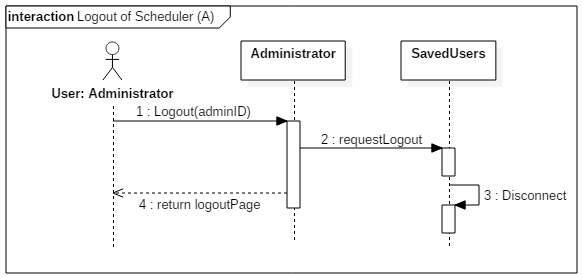
*Figure 5.2.1 Administrator logout system sequence diagram*



*Figure 5.2.2 Student logout system sequence diagram*

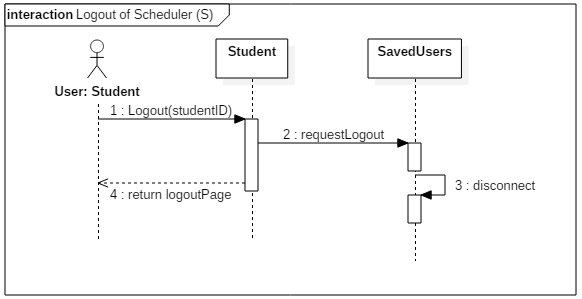
Contract Diagrams

|  |  |  |
| --- | --- | --- |
| Contract 2.1 | Logout | |
| Created By: | Ryan Lee | Last Updated By: Ryan Lee |
| Date Created: | March 17, 2016 | Last Revision Date: March 17, 2016 |
| Operation: | logout(adminID: long) | |
| Cross-reference: | UC2 | |
| Preconditions: | * The user is logged into their account. | |
| Post-conditions: | * The user is disconnected from the server and logged out. (instance deletion) | |



*Figure 5.2.3 Administrator logout contract diagram*

|  |  |  |
| --- | --- | --- |
| Contract 2.2 | Logout | |
| Created By: | Ryan Lee | Last Updated By: Ryan Lee |
| Date Created: | March 17, 2016 | Last Revision Date: March 17, 2016 |
| Operation: | logout(studentID: long) | |
| Cross-reference: | UC2 | |
| Preconditions: | * The user is logged into their account. | |
| Post-conditions: | * The user is disconnected from the server and logged out. (instance deletion) | |



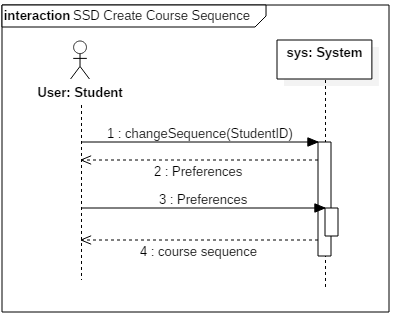
*Figure 5.2.4 Student logout contract diagram*

Use Case 3: Create Course Sequence

Fully Dressed Use Case

|  |  |  |
| --- | --- | --- |
| Use Case ID: | UC3 | |
| Use Case Name: | Create Course Sequence | |
| Created By: | Marc-Andre Leclair | Last Updated By: Ryan Lee |
| Date Created: | January 25th, 2016 | Last Revision Date: March 9, 2016 |
| Actor(s): | Student | |
| Goal/Actor Goals: | A student wants to create a course sequence | |
| Description/Summary: | The Student wants to create his or her course sequence. The system must provide the user with the option to add their own preferences in order to generate a sequence that is personalized to these preferences. | |
| Preconditions: | * The user is logged into the system * The user is registered in a program. | |
| Post-conditions: | The user has a personalized course sequence for the future | |
| Minimum Guarantee: | No course sequence will be created | |
| Basic Flow: | 1. Student requests to create a sequence. 2. The system must prompt the student to add his or her preferences to the sequence. 3. The student selects zero, one, or more preferences. 4. The system prompts the student to confirm their preferences 5. The student confirms his choices. 6. The course sequence is created and shown to the student. | |
| Risk assessment: | High | |
| Importance assessment: | 5/5 | |

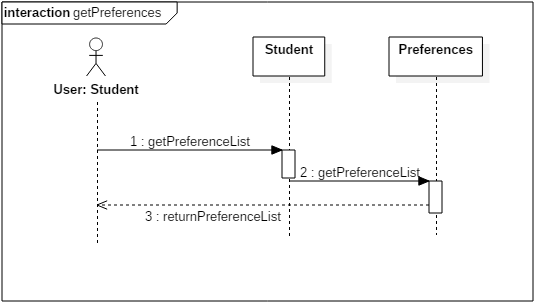
System Sequence Diagram



*Figure 5.3.1 Create course system sequence diagram*

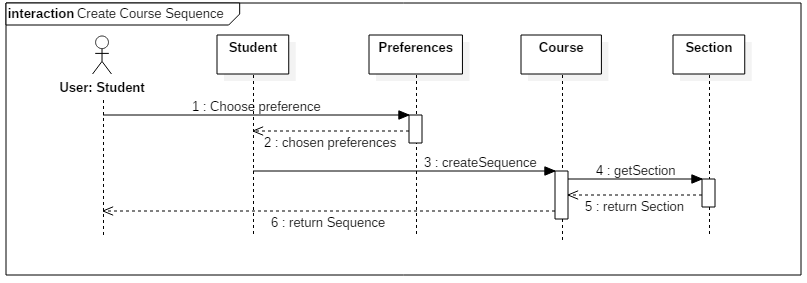
Contract Diagrams

|  |  |  |
| --- | --- | --- |
| Contract 3.1 | getPreference | |
| Created By: | Ryan Lee | Last Updated By: Ryan Lee |
| Date Created: | March 17, 2016 | Last Revision Date: March 17, 2016 |
| Operation: | getPreferences( ) | |
| Cross-reference: | UC3 | |
| Preconditions: | * The user is logged into the system * The user is registered in a program. | |
| Post-conditions: | none | |



*Figure 5.3.2 getPreferences contract diagram*

|  |  |  |
| --- | --- | --- |
| Contract 3.2 | choosePreferences | |
| Created By: | Ryan Lee | Last Updated By: Ryan Lee |
| Date Created: | March 17, 2016 | Last Revision Date: March 17, 2016 |
| Operation: | choosePreferences() | |
| Cross-reference: | UC3 | |
| Preconditions: | * The user is logged into the system * The user is registered in a program. | |
| Post-conditions: | * The user has a personal sequence created for them. (attribute modification, association creation) | |



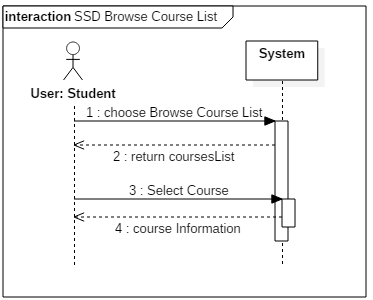
*Figure 5.3.3 createCourseSequence contract diagram*

Use Case 4: Browse Course List

Fully Dressed Use Case

|  |  |  |
| --- | --- | --- |
| Use Case ID: | UC4 | |
| Use Case Name: | Browse Course List | |
| Created By: | Ideawin-Bunthy Koun | Last Updated By: Ryan Lee |
| Date Created: | January 25th, 2016 | Last Revision Date: March 9, 2016 |
| Actor(s): | Student | |
| Goal/Actor Goals: | Browse the list of courses from the course calendar. | |
| Description/Summary: | The user wishes to view available courses and access information pertaining to them. The system must display such information, including sections, times, and locations. | |
| Preconditions: | * User must be logged in as a Student. * Course list must be available. | |
| Post-conditions: | A list of courses is displayed. | |
| Minimum Guarantee: | The system fails to display a list of courses and displays an error message to the user. | |
| Basic Flow: | 1. Student selects the ‘Browse Course List” feature. 2. System retrieves list of courses. 3. System displays list of courses. 4. User selects a course from that list. 5. System displays information about the selected course. | |
| Risk assessment: | Medium | |
| Importance assessment: | 3/5 | |

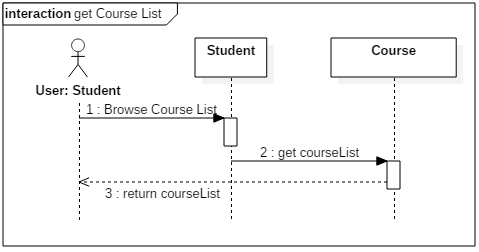
System Sequence Diagram



*Figure 5.4.1 Browse course list system sequence diagram*

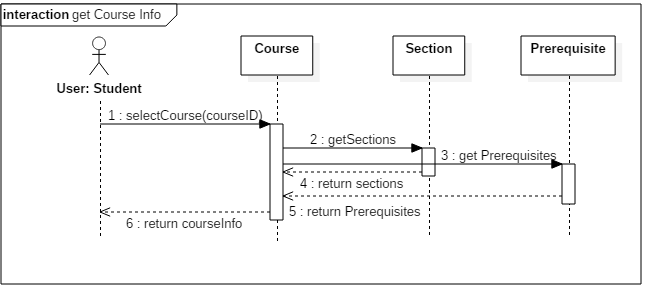
Contract Diagrams

|  |  |  |
| --- | --- | --- |
| Contract 4.1 | browseCourseList | |
| Created By: | Ryan Lee | Last Updated By: Ryan Lee |
| Date Created: | March 17, 2016 | Last Revision Date: March 17, 2016 |
| Operation: | getCourseList( ) | |
| Cross-reference: | UC4 | |
| Preconditions: | * User must be logged in as a Student. * Course list must be available. | |
| Post-conditions: | None | |



*Figure 5.4.2 getCourseList contract diagram*

|  |  |  |
| --- | --- | --- |
| Contract 4.2 | selectCourse | |
| Created By: | Ryan Lee | Last Updated By: Ryan Lee |
| Date Created: | March 17, 2016 | Last Revision Date: March 17, 2016 |
| Operation: | selectCourse(courseID: long) | |
| Cross-reference: | UC4 | |
| Preconditions: | * User must be logged in as a Student. * Course list must be available. | |
| Post-conditions: | None | |



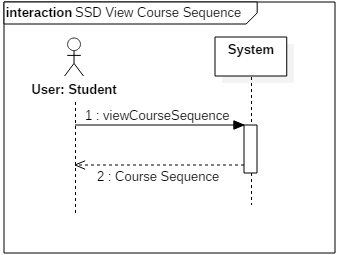
*Figure 5.4.3 getCourseInfo contract diagram*

Use Case 5: View Course Sequence

Fully Dressed Use Case

|  |  |  |
| --- | --- | --- |
| Use Case ID: | UC5 | |
| Use Case Name: | View Course Sequence | |
| Created By: | Ideawin-Bunthy Koun | Last Updated By: Ryan Lee |
| Date Created: | January 25th, 2016 | Last Revision Date: March 9, 2016 |
| Actor(s): | Student | |
| Goal/Actor Goals: | Allows the user to view their expected progression in the program. | |
| Description/Summary: | User can view a complete course sequence of the program the student is enrolled in. The Sequence will show the courses the Student is expected to take, and the order he or she is expected to take them in. | |
| Preconditions: | * User must be logged in as a Student * The system has access to a course sequence. | |
| Post-conditions: | The system displays a course sequence. | |
| Minimum Guarantee: | The system fails to display a course sequence and displays an error message to the user. | |
| Basic Flow: | 1. Student selects the option to view his or her course sequence. 2. System retrieves the sequence. 3. Course sequence is displayed to the student. | |
| Risk assessment: | Medium | |
| Importance assessment: | 4/5 | |

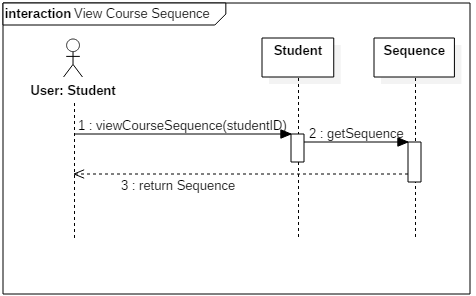
System Sequence Diagram



*Figure 5.5.1 View Course Sequence system sequence diagram*

Contract diagrams

|  |  |  |
| --- | --- | --- |
| Contract 5.1 | viewCourseSequence | |
| Created By: | Ryan Lee | Last Updated By: Ryan Lee |
| Date Created: | March 17, 2016 | Last Revision Date: March 17, 2016 |
| Operation: | viewCourseSequence(studentID: long) | |
| Cross-reference: | UC5 | |
| Preconditions: | * User must be logged in as a Student * The system has access to a course sequence. | |
| Post-conditions: | None | |



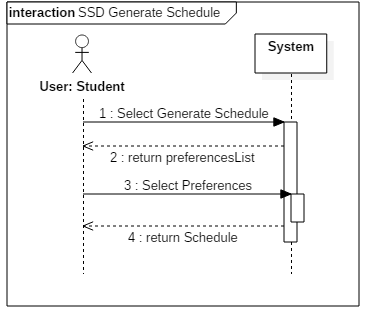
*Figure 5.5.2 viewCourseSequence contract diagram*

Use Case 6: Generate Schedule

Fully Dressed Use Case

|  |  |  |
| --- | --- | --- |
| Use Case ID: | UC6 | |
| Use Case Name: | Generate Schedule | |
| Created By: | Ideawin-Bunthy Koun | Last Updated By: Ryan Lee |
| Date Created: | January 25th, 2016 | Last Revision Date: March 9, 2016 |
| Actor(s): | Student | |
| Goal/Actor Goals: | Obtain a suggested schedule for the next four years. | |
| Description/Summary: | The system can generate a 4-year schedule for the user based on the student’s preferences and constraints and on prerequisite and co-requisite policy. | |
| Preconditions: | * User must be logged in as a Student. * The system must have access to course database. * The system must have access to the student’s records. | |
| Post-conditions: | The system generates a schedule with no overlaps. | |
| Minimum Guarantee: | The system fails to generate a schedule and displays an error message to the user. | |
| Basic Flow: | 1. User selects the ‘Generate Schedule’ feature. 2. System retrieves list of preferences. 3. User selects his or her options and preferences. 4. System prompts user to confirm preferences. 5. System responds by displaying a schedule that meets the user’s preferences and constraints. | |
| Risk assessment: | High | |
| Importance assessment: | 5/5 | |

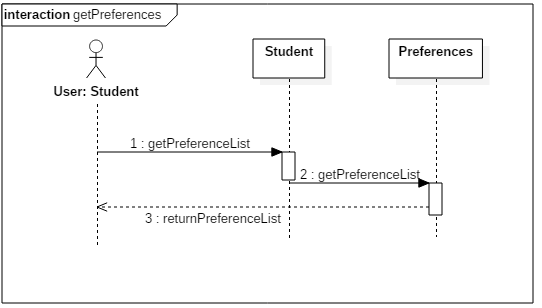
System Sequence Diagram



*Figure 5.6.1 Generate Schedule System Sequence Diagram*

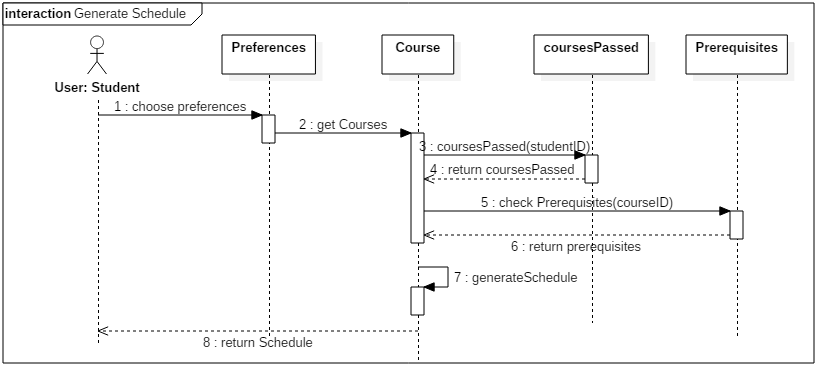
Contract Diagrams

|  |  |  |
| --- | --- | --- |
| Contract 6.1 | getPreferenceList | |
| Created By: | Ryan Lee | Last Updated By: Ryan Lee |
| Date Created: | March 17, 2016 | Last Revision Date: March 17, 2016 |
| Operation: | getPreferenceList() | |
| Cross-reference: | UC6 | |
| Preconditions: | * User must be logged in as a Student. * The system must have access to course database. * The system must have access to the student’s records. | |
| Post-conditions: | None | |



*Figure 5.6.2 getPreferences contract diagram*

|  |  |  |
| --- | --- | --- |
| Contract 6.2 | choosePreferences | |
| Created By: | Ryan Lee | Last Updated By: Ryan Lee |
| Date Created: | March 17, 2016 | Last Revision Date: March 17, 2016 |
| Operation: | choosePreferences (studentID: long) | |
| Cross-reference: | UC6 | |
| Preconditions: | * User must be logged in as a Student. * The system must have access to course database. * The system must have access to the student’s records. | |
| Post-conditions: | * An instance of Schedule, schd, will be created (instance creation) * Schd will be added to student’s saved schedules. (attribute modification, association creation) | |



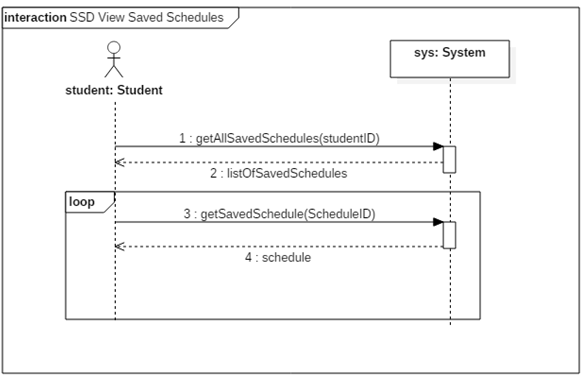
*Figure 5.6.3 generateSchedule contract diagram*

Use Case 7 – View Saved Schedules

Fully Dressed Use Case

|  |  |  |
| --- | --- | --- |
| Use Case ID: | UC7 | |
| Use Case Name: | View Saved Schedules | |
| Created By: | Erin Benderoff | Last Updated By: Claudia Della Serra |
| Date Created: | February 7, 2016 | Last Revision Date: February 14, 2016 |
| Actor(s): | Student | |
| Goal/Actor Goals: | The user wishes to view a previously saved schedule. | |
| Description/Summary: | The user selects a schedule from a list of their previously saved schedules. The system must display this schedule for the user to review. | |
| Preconditions: | * User is logged into the system. * User has previously generated one or more schedules * User has saved at least one generated schedule | |
| Post-conditions: | The system displays the saved schedule | |
| Minimum Guarantee: | Saved schedules remain in the system unchanged. | |
| Basic Flow: | 1. The user indicates their wish to view a saved schedule 2. The system prompts the user with a list of saved schedules 3. The user selects which semester’s schedule to view 4. The system displays the chosen schedule | |
| Risk assessment: | Medium | |
| Importance assessment: | 3/5 | |

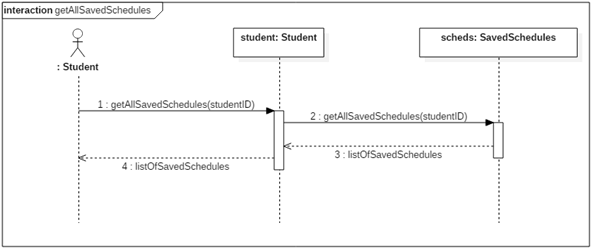
System Sequence Diagram



*Figure 5.7.1 View Saved Schedules system sequence diagram*

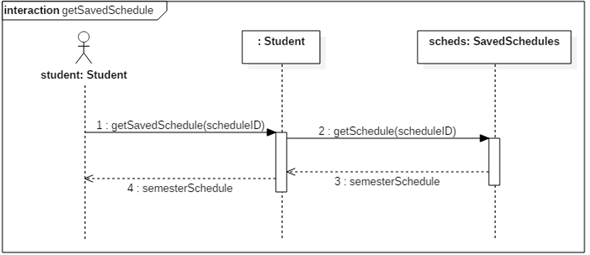
Contract Diagrams

|  |  |  |
| --- | --- | --- |
| Contract 7.1 | getAllSavedSchedules | |
| Created By: | Claudia Della Serra | Last Updated By: Claudia Della Serra |
| Date Created: | March 2, 2016 | Last Revision Date: March 2, 2016 |
| Operation: | viewSavedSchedules(studentID: long) : list<SavedSchedule> | |
| Cross-reference: | UC7 | |
| Preconditions: | * User is logged into the system. * User has previously generated one or more schedules * User has saved at least one generated schedule | |
| Post-conditions: | None | |



*Figure 5.7.2 getAllSavedSchedules contract diagrams*

|  |  |  |
| --- | --- | --- |
| Contract 7.2 | getSavedSchedule | |
| Created By: | Claudia Della Serra | Last Updated By: Claudia Della Serra |
| Date Created: | March 2, 2016 | Last Revision Date: March 2, 2016 |
| Operation: | getSavedSchedule(scheduleID: long) : Schedule | |
| Cross-reference: | UC7 | |
| Preconditions: | * User is logged into the system. * User has previously generated one or more schedules * User has saved at least one generated schedule | |
| Post-conditions: | None | |



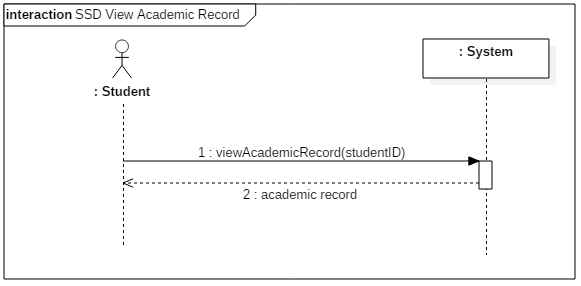
*Figure 5.7.3 getSavedSchedule contract diagram*

Use Case 8 – View Academic Record

Fully Dressed Use Case

|  |  |  |
| --- | --- | --- |
| Use Case ID: | UC8 | |
| Use Case Name: | View Academic Record | |
| Created By: | Erin Benderoff | Last Updated By: Claudia Della Serra |
| Date Created: | February 7, 2016 | Last Revision Date: February 14, 2016 |
| Actor(s): | Student | |
| Goal/Actor Goals: | The user wishes to view a list of his or her completed courses and grades for those courses. | |
| Description/Summary: | The system displays the user’s academic record, which includes completed courses and courses currently being taken. | |
| Preconditions: | * User is logged into the system. * User has completed at least one course and/or is presently enrolled in at least one course. | |
| Post-conditions: | The system displays the user’s academic record. | |
| Minimum Guarantee: | The academic record remains in the system unmodified. | |
| Basic Flow: | 1. The user requests to view their academic record 2. The system generates a list of the user’s completed courses, as well as courses in which the user is currently enrolled. | |
| Risk assessment: | Low | |
| Importance assessment: | 1/5 | |

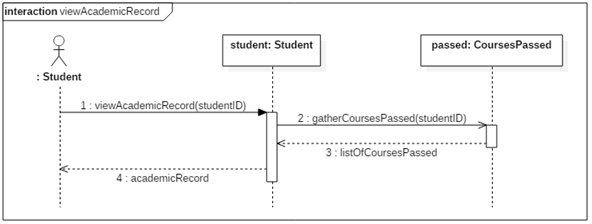
System Sequence Diagram



*Figure 5.8.1 View Academic Record System Sequence Diagram*

Contract Diagrams

|  |  |  |
| --- | --- | --- |
| Contract 8.1 | viewAcademicRecord | |
| Created By: | Claudia Della Serra | Last Updated By: Claudia Della Serra |
| Date Created: | March 2, 2016 | Last Revision Date: March 2, 2016 |
| Operation: | viewAcademicRecord (studentID: long) : list<Course> | |
| Cross-reference: | UC8 | |
| Preconditions: | * User is logged into the system. * User has completed or is presently enrolled in at least one course | |
| Post-conditions: | * A list of classes passedis dynamically created for the student (instance creation) | |



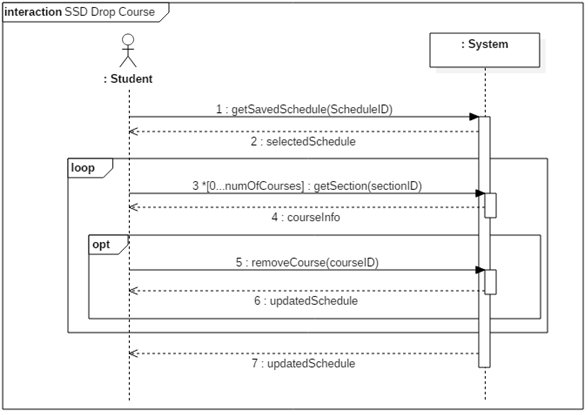
*Figure 5.8.2 viewAcademicRecord contract diagram*

Use Case 9 – Drop Course

Fully Dressed Use Case

|  |  |  |
| --- | --- | --- |
| Use Case ID: | UC9 | |
| Use Case Name: | Drop Course | |
| Created By: | Erin Benderoff | Last Updated By: Claudia Della Serra |
| Date Created: | February 7, 2016 | Last Revision Date: February 14, 2016 |
| Actor(s): | Student | |
| Goal/Actor Goals: | The user wishes to remove a course from a generated schedule. | |
| Description/Summary: | The user selects which course he or she would like to remove from the schedule. The system must delete this course from the schedule. | |
| Preconditions: | * User is logged into the system. * User has generated a schedule containing at least one course. * User has chosen to view a saved schedule | |
| Post-conditions: | The chosen course has been removed from the user’s schedule. | |
| Minimum Guarantee: | The courses on the user’s schedule remain unchanged. | |
| Basic Flow: | 1. User selects a schedule they wish to view 2. System returns the desired Schedule 3. The user chooses a course from his or her schedule to view. 4. The system displays the course information 5. The user indicates they wish to remove the course from the schedule 6. The System prompts the user for confirmation. 7. The user confirms dropping of the course 8. The produces a modified schedule without the removed course. | |
| Risk assessment: | High | |
| Importance assessment: | 4/5 | |

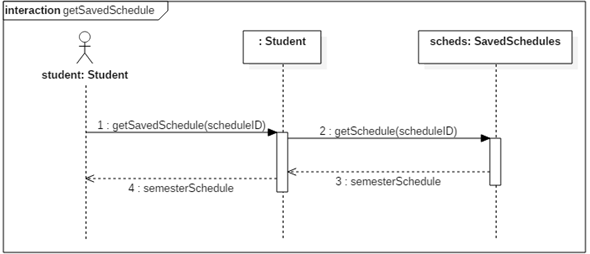
System Sequence Diagram



*Figure 5.9.1 Drop Course system sequence diagram*

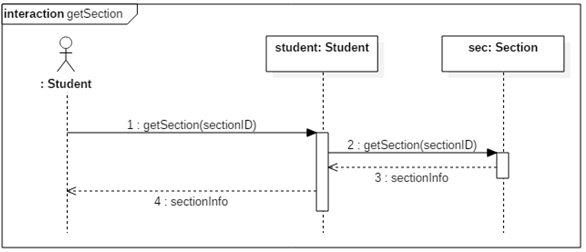
Contract Diagrams

|  |  |  |
| --- | --- | --- |
| Contract 9.1 | getSavedSchedule | |
| Created By: | Claudia Della Serra | Last Updated By: Claudia Della Serra |
| Date Created: | March 2, 2016 | Last Revision Date: March 2, 2016 |
| Operation: | getSavedSchedule(scheduleID: long) | |
| Cross-reference: | UC9, UC7, Contract 7.2 | |
| Preconditions: | * User is logged into the system. * User has generated and saved a schedule * User has requested to view a list of saved schedules | |
| Post-conditions: | None | |



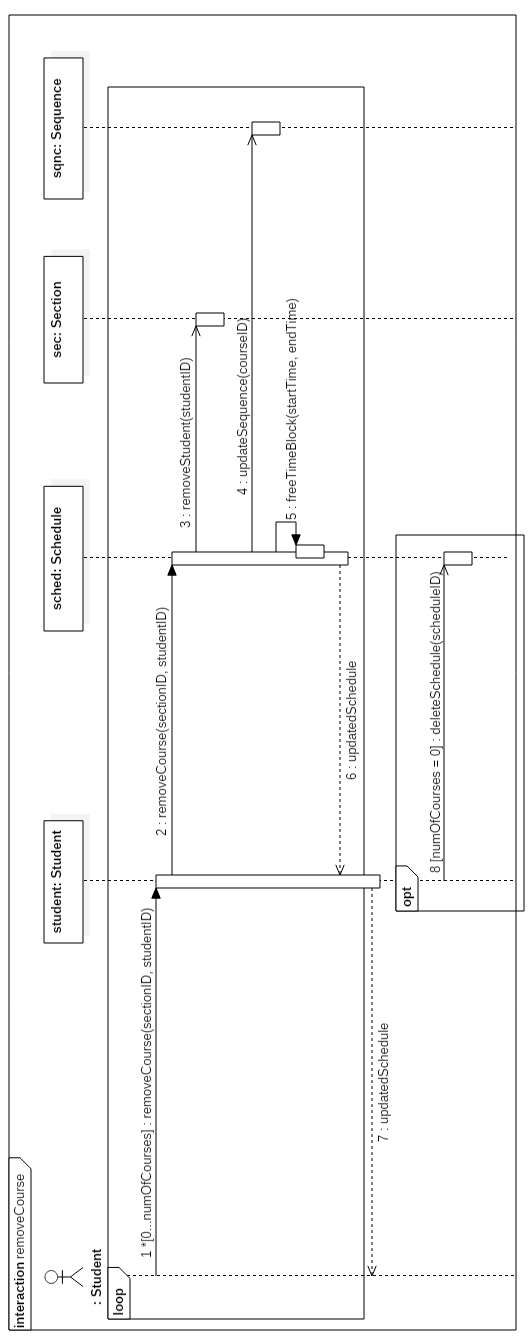
*Figure 5.9.2 getSavedSchedule contract diagram*

|  |  |  |
| --- | --- | --- |
| Contract 9.2 | getSection | |
| Created By: | Claudia Della Serra | Last Updated By: Claudia Della Serra |
| Date Created: | March 2, 2016 | Last Revision Date: March 2, 2016 |
| Operation: | viewCourseInfo(sectionID: long) : Section | |
| Cross-reference: | UC9 | |
| Preconditions: | * User is logged into the system. * User has generated and saved a schedule * User has requested to view a saved schedule | |
| Post-conditions: | None | |



*Figure 5.9.3 getSection contract diagram*

|  |  |  |
| --- | --- | --- |
| Contract 9.3 | removeCourse | |
| Created By: | Claudia Della Serra | Last Updated By: Claudia Della Serra |
| Date Created: | March 2, 2016 | Last Revision Date: March 2, 2016 |
| Operation: | removeCourse(sectionID: long, studentID: long) : Schedule | |
| Cross-reference: | UC9 | |
| Preconditions: | * User is logged into the system. * User has generated and saved a schedule * User has requested to view a saved schedule * User has requested to view course registration info | |
| Post-conditions: | * The student will be removed from the section’s registration list (attribute modification, association removed) * The course will be removed from the student’s schedule (association modification) * The student’s course sequence will be updated (attribute modification) | |



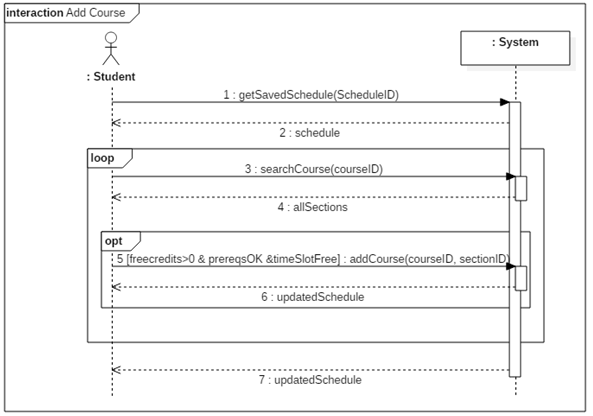
*Figure 5.9.4 removeCourse contract diagram*

Use Case 10 – Add Course

Fully Dressed Use Case

|  |  |  |
| --- | --- | --- |
| Use Case ID: | UC10 | |
| Use Case Name: | Add Course | |
| Created By: | Lori Dalkin | Last Updated By: Claudia Della Serra |
| Date Created: | February 7, 2016 | Last Revision Date: February 14, 2016 |
| Actor(s): | Student | |
| Goal/Actor Goals: | Add a course to a generated schedule. | |
| Description/Summary: | Specific courses can be added to a schedule generated by a user. The user must search for the course and manually add it to their schedule. | |
| Preconditions: | * The user is logged in to their account * A schedule has been generated | |
| Post-conditions: | The specified course is added to the schedule. | |
| Minimum Guarantee: | The user’s schedule remains unchanged and no new course is added | |
| Basic Flow: | 1. User requests to view schedule. 2. System displays the indicated schedule. 3. User indicates they wish to add a course 4. The system displays a search screen prompting the user to enter the name of the course 5. The user types in the name of the course they wish to add 6. The system displays a list of corresponding courses and sections 7. User selects the specific section of the course they wish to add. 8. The system prompts the user to confirm their selected course 9. The user confirms their wish to add the course 10. The system returns a modified schedule. | |
| Risk assessment: | Medium | |
| Importance assessment: | 3/5 | |

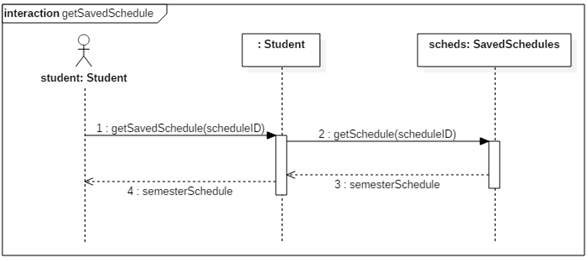
System Sequence Diagram



*Figure 5.10.1 Add Course system sequence diagram*

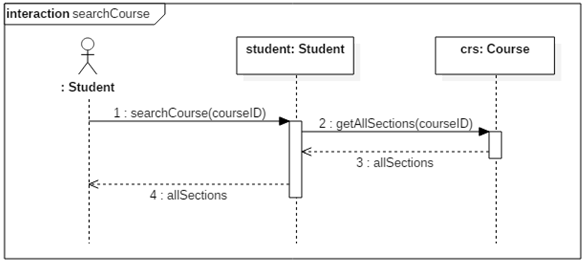
Contract Diagrams

|  |  |  |
| --- | --- | --- |
| Contract 10.1 | getSavedSchedule | |
| Created By: | Claudia Della Serra | Last Updated By: Claudia Della Serra |
| Date Created: | March 2, 2016 | Last Revision Date: March 2, 2016 |
| Operation: | removeCourse(courseID: long, sectionID: long) | |
| Cross-reference: | UC10, UC7, Contract 7.2 | |
| Preconditions: | * User is logged into the system. * User has generated and saved a schedule * User has requested to view a saved schedule | |
| Post-conditions: | None | |



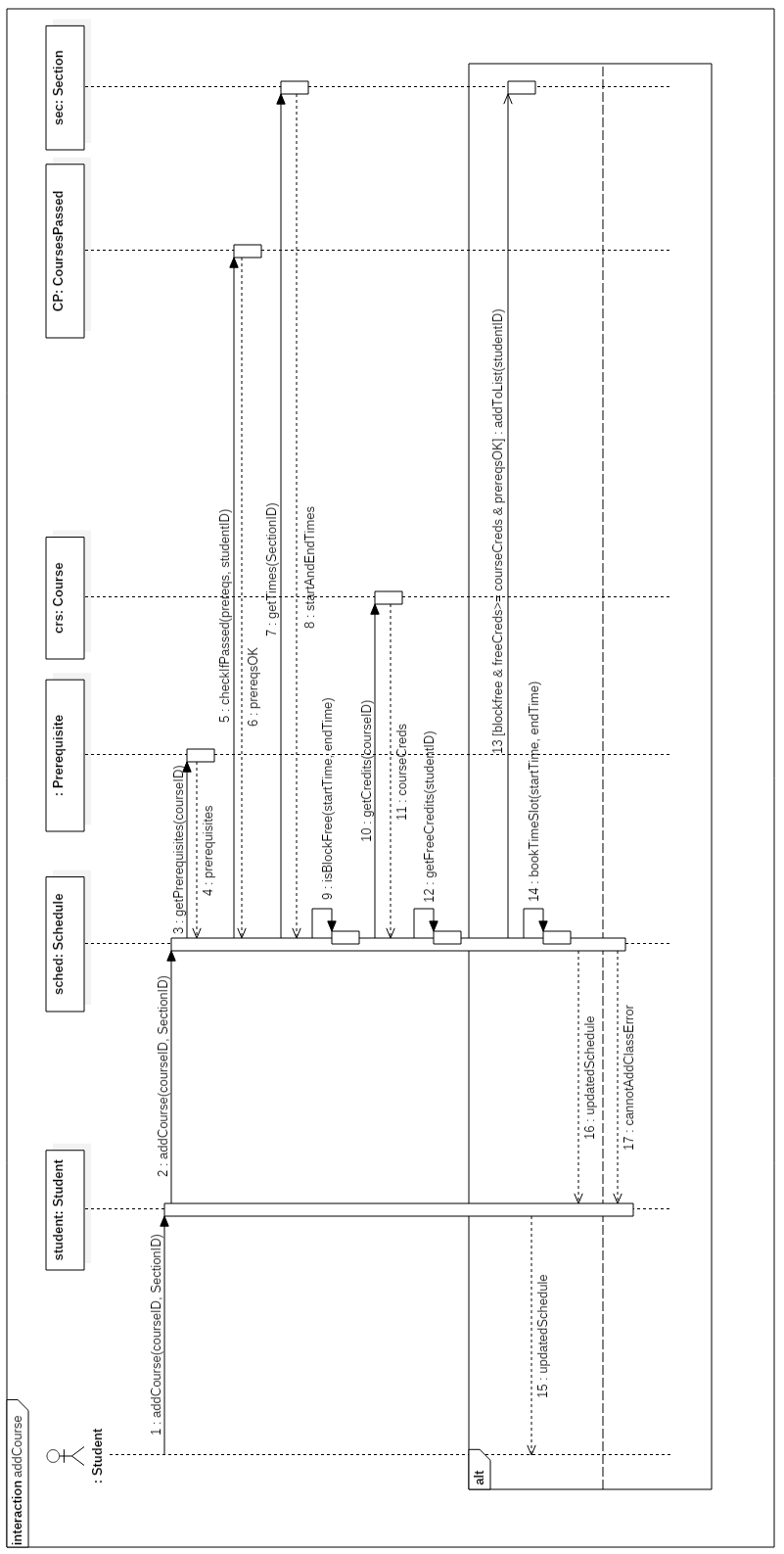
*Figure 5.10.2 getSavedSchedule contract diagram*

|  |  |  |
| --- | --- | --- |
| Contract 10.2 | searchCourse | |
| Created By: | Claudia Della Serra | Last Updated By: Claudia Della Serra |
| Date Created: | March 2, 2016 | Last Revision Date: March 2, 2016 |
| Operation: | searchCourse(courseID) : list<Section> | |
| Cross-reference: | UC10 | |
| Preconditions: | * User is logged into the system. * User has generated and saved a schedule * User has requested to view a saved schedule | |
| Post-conditions: | None | |



*Figure 5.10.3 searchCourse contract diagram*

|  |  |  |
| --- | --- | --- |
| Contract 10.3 | addCourse | |
| Created By: | Claudia Della Serra | Last Updated By: Claudia Della Serra |
| Date Created: | March 2, 2016 | Last Revision Date: March 2, 2016 |
| Operation: | addCourse (courseID, sectionID) | |
| Cross-reference: | UC10 | |
| Preconditions: | * User is logged into the system. * User has generated and saved a schedule * User has requested to view a saved schedule * User has enough free credits to add a class * User has completed the required prerequisites | |
| Post-conditions: | * The student will be added to the section’s registration list (attribute modification, association formed) * The class will be added to the student’s schedule (association formed) | |



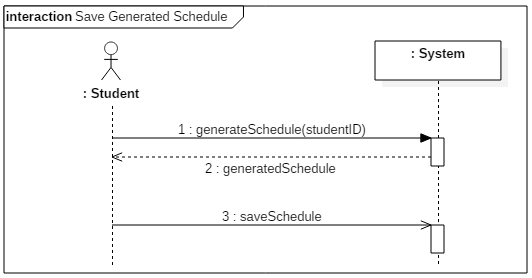
*Figure 5.10.4 addCourse contract diagram*

Use Case 11 – Save Generated Schedule

Fully Dressed Use Case

|  |  |  |
| --- | --- | --- |
| Use Case ID: | UC11 | |
| Use Case Name: | Save Generated Schedule | |
| Created By: | Lori Dalkin | Last Updated By: Claudia Della Serra |
| Date Created: | February 7, 2016 | Last Revision Date: February 14, 2016 |
| Actor(s): | Student | |
| Goal/Actor Goals: | The user wishes to save a schedule they have generated. | |
| Description/Summary: | A generated schedule can be saved in order to be accessed and viewed later on by the user. This schedule consists of courses the user is satisfied with and wishes to keep on their schedule. | |
| Preconditions: | * The user is logged on to their account * A schedule has been generated containing at least one course. | |
| Post-conditions: | The system saves the generated schedule to the user’s account. | |
| Minimum Guarantee: | The user’s account remains unchanged. | |
| Basic Flow: | 1. User indicated their wish to generate a schedule 2. The system prompts the user with a list of semesters for which they wish to generate a schedule 3. The user indicates which semester they wish to generate a schedule for 4. The system displays a generated schedule for the indicated semester 5. The user will indicates that they want to save the generated schedule. 6. The system will display a message telling the user that the schedule has been saved. | |
| Risk assessment: | Medium | |
| Importance: | 4/5 | |

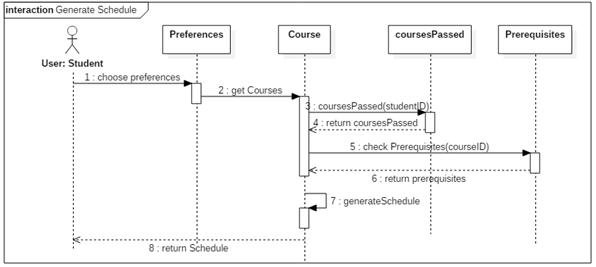
System Sequence Diagram



*Figure 5.11.1 Save Generated Schedule system sequence diagram*

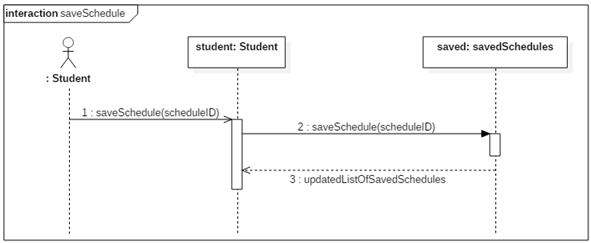
Contract Diagrams

|  |  |  |
| --- | --- | --- |
| Contract 11.1 | generateSchedule | |
| Created By: | Claudia Della Serra | Last Updated By: Claudia Della Serra |
| Date Created: | March 2, 2016 | Last Revision Date: March 2, 2016 |
| Operation: | generateSchedule(studentID) | |
| Cross-reference: | UC11, UC6, contract 6.2 | |
| Preconditions: | * User is logged into the system. * User is enrolled in a program | |
| Postconditions: | None | |



*Figure 5.11.2 generateSchedule contract diagram*

|  |  |  |
| --- | --- | --- |
| Contract 11.2 | saveGeneratedSchedule | |
| Created By: | Claudia Della Serra | Last Updated By: Claudia Della Serra |
| Date Created: | March 2, 2016 | Last Revision Date: March 2, 2016 |
| Operation: | saveGeneratedSchedule(scheduleID): list<Schedule> | |
| Cross-reference: | UC11 | |
| Preconditions: | * User is logged into the system. * User is enrolled in a program * User has generated a schedule | |
| Post-conditions: | * A schedule for that semester will be saved to the student’s account (association formed) * The schedule will be added to the list of saved schedules (attribute modification) | |



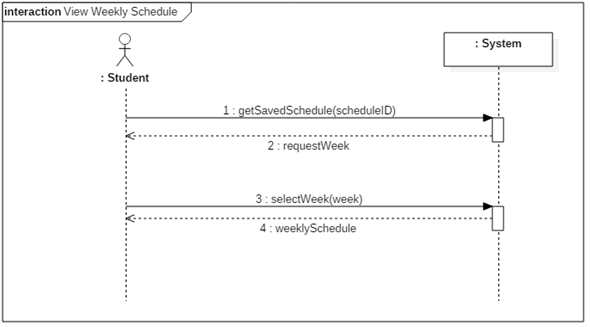
*Figure 5.11.3 saveSchedule contract diagram*

Use Case 12 – View Weekly Schedule

Fully Dressed Use Case

|  |  |  |
| --- | --- | --- |
| Use Case ID: | UC12 | |
| Use Case Name: | View Weekly Schedule | |
| Created By: | Bryce Drewery Schoeler | Last Updated By: Claudia Della Serra |
| Date Created: | February 7, 2016 | Last Revision Date: February 14, 2016 |
| Actor(s): | Student | |
| Goal/Actor Goals: | A student wants to view weekly schedule. | |
| Description/Summary: | The student wishes to see their class schedule for the current week. The student will specify a week to be displayed. The system must display the schedule for that week. | |
| Preconditions: | * The student has been authenticated * A schedule has been generated and saved | |
| Post-conditions: | The schedule is shown on screen. | |
| Minimum Guarantee: | The schedule fails to be displayed. | |
| Basic Flow: | 1. Student requests to see a saved schedule. 2. The system prompts the student to select a week. 3. Student selects a week 4. The systems display the schedule for that week | |
| Risk assessment: | Medium | |
| Importance assessment: | 3/5 | |

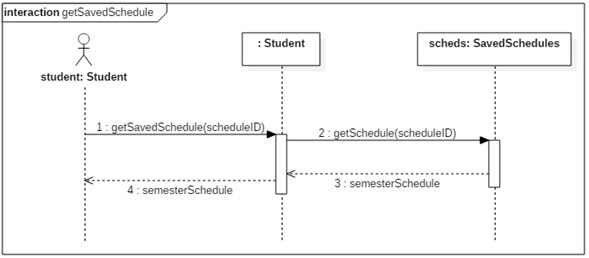
System Sequence Diagram



*Figure 5.12.1 View Weekly Schedule system sequence diagram*

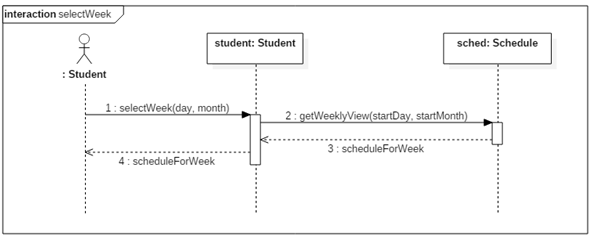
Contract Diagrams

|  |  |  |
| --- | --- | --- |
| Contract 12.1 | getSavedSchedule | |
| Created By: | Claudia Della Serra | Last Updated By: Claudia Della Serra |
| Date Created: | March 2, 2016 | Last Revision Date: March 2, 2016 |
| Operation: | getSavedSchedule(scheduleID) | |
| Cross-reference: | UC12, UC7, contract 7.2 | |
| Preconditions: | * User is logged into the system. * User is enrolled in a program * User has saved a schedule | |
| Post-conditions: | None | |



*Figure 5.12.2 getSavedSchedule contract diagram*

|  |  |  |
| --- | --- | --- |
| Contract 12.2 | selectWeekSchedule | |
| Created By: | Claudia Della Serra | Last Updated By: Claudia Della Serra |
| Date Created: | March 2, 2016 | Last Revision Date: March 2, 2016 |
| Operation: | selectWeek(date: int, month: int) | |
| Cross-reference: | UC12, | |
| Preconditions: | * User is logged into the system. * User is enrolled in a program * User has saved a schedule | |
| Post-conditions: | None | |



*Figure 5.12.3 selectWeek Contract diagram*

# Estimation

The current cost and time estimation has been updated from previous estimations made in deliverable 1 by considering the estimated lines of code that will be produced in the implementation of the TimeTurner system. An estimated person-months amount was calculated, and then this amount applied to previous estimations made in conjunction with estimations based on time-logs taken from activities performed in the production of the Deliverable 2 document.

Function Point Estimation

Unadjusted Function Points

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Function Type** | | **Functional Complexity** | | | | | |  | | **Complexity Totals** | | | | | | **Function Type Totals** | |
| ILF | | 0 Low | | | | | | X 7 | | 0 | | | | | |  | |
|  | | 10 Average | | | | | | X10 | | 100 | | | | | |  | |
|  | | 0 High | | | | | | X15 | | 0 | | | | | | 100 | |
|  | | | | | | | | | | | | | | | | | |
| EIF | | | | 0 Low | | | X 5 | | 0 | | | | | |  | | |
|  | | | | 0 Average | | | X7 | | 0 | | | | | |  | | |
|  | | | | 0 High | | | X10 | | 0 | | | | | | 0 | | |
|  | | | | | | | | | | | | | | | | | |
| EI | | | | | 0 Low | | | X3 | | | 0 | | | | |  | |
|  | | | | | 4 Average | | | X4 | | | 16 | | | | |  | |
|  | | | | | 0 High | | | X6 | | | 0 | | | | | 16 | |
|  | | | | | | | | | | | | | | | | | |
| EQ | | | 0 Low | | | | X 3 | | | | | | 0 |  | | | |
|  | | | 0 Average | | | | X4 | | | | | | 0 |  | | | |
|  | | | 6 High | | | | X6 | | | | | | 36 | 36 | | | |
|  | | | | | | | | | | | | | | | | | |
| EO | 0 Low | | | | | X 4 | | | | | | 0 | |  | | | |
|  | 5 Average | | | | | X5 | | | | | | 25 | |  | | | |
|  | 0 High | | | | | X7 | | | | | | 0 | | 25 | | | |
|  | | | | | | | | | | | | | | | | | |
| **Unadjusted Function Point Count** | | | | | | | | | | | | | | | | | 177 |
| **SLOC/FP for PHP [1]** | | | | | | | | | | | | | | | | | 67 |
| **Total SLOC** | | | | | | | | | | | | | | | | | 11859 |
| **COCOMO II estimated person-months effort** | | | | | | | | | | | | | | | | | 43.8 |

[1] taken from <https://www.cs.helsinki.fi/u/taina/ohtu/fp.html>

Internal function points were calculated based on the database tables given for Users, User Schedules, Completed Courses, Courses, Sections, Subsections, and Prerequisites. External Outputs, External Queries and External Inputs were calculated based on methods found in the above sections (see Section 3.2 Subsystem Interface Specification). Given the above estimated lines of code, this value was put into the COCOMO II engine, with the Software Scale Drivers modified as follows:

* Pecedentedness: Low
  + Due to the fact that not very much experience in programming a large system such as this one has been previously incurred by many of the team members.
* Architecture/Risk Resolution: high
  + Due to the fact that much planning in architecture and design has been undergone, and that risk assessment and analysis has been performed since the start of the project.
* Process Maturity: low
  + Due to the fact that the team has only begun using software process management, and does not have much experience in handling projects in this manner
* Development flexibility: high
  + Due to the fact that very vague requirements were given by instructors, thus the flexibility to implement these requirements, and create our own, was very high.
* Team cohesion: high
  + The team has had much experience working together on previous assignments, labs, etc. thus team cohesion is high and group work and communication are excellent.

Module Estimations

Given the estimated time of 43.8 person-months, and previous time estimates that were performed in the first version of the project, new time estimates have been made. Given that all team members are currently full-time students and working on other projects throughout the course of this project development, our person-months are based on a rough estimate of 4.85 hours per person per task; such an estimate is taken from the time logs of one team member working on the Deliverable 2 task of creating system sequence and contract diagrams (times were divided by 2 due to the fact that all system sequence diagrams were implemented rather than just 2 major ones). This results in a 424.86 person-hours estimate for the entire project. Thus, previous estimates are updated as follows:

6.3.4 Total Deliverable Estimates

|  |  |  |
| --- | --- | --- |
| **Deliverable Names** | **Cost (Hours)** | **Revised Cost (Hours)** |
| Deliverable 0 | 29.5 | 29.5 |
| Deliverable 1 | 35 | 35 |
| Deliverable 2 | 190 | 194 |
| Deliverable 3 | 75 | 97 |
| Deliverable 4 | 60 | 48.5 |
| **Total** | **389.5** | **404** |

Updated Gantt Chart

The following updated version of the initial Gantt chart shows the progress of the project and its current state of development.

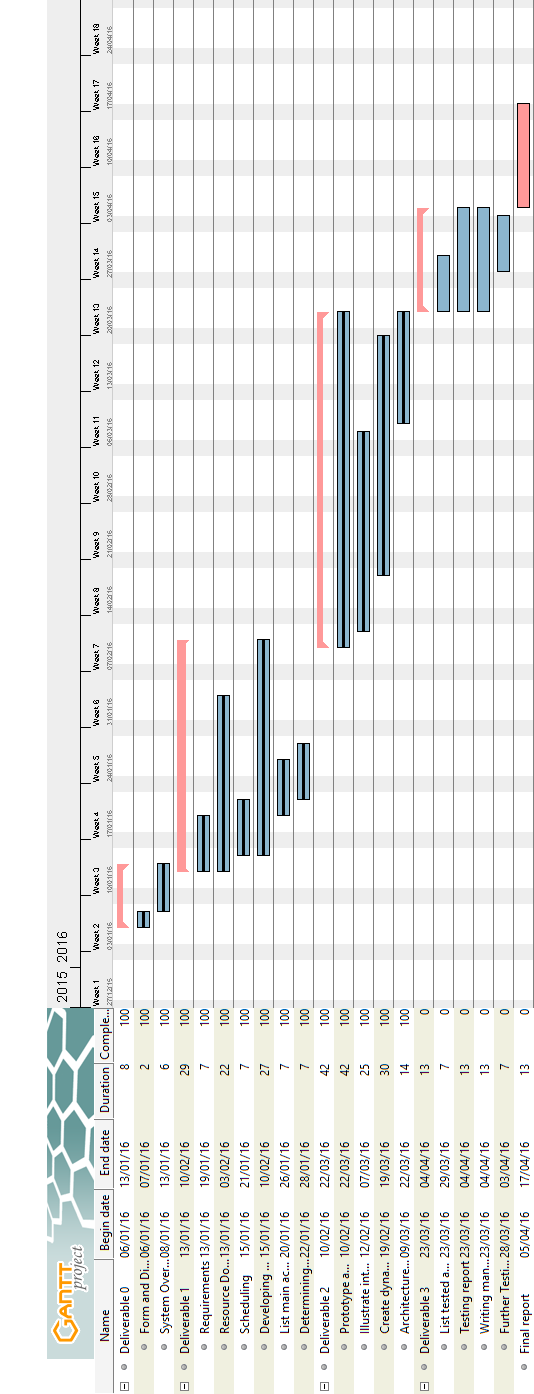


Figure 6.3.1 Updated Gantt Chart

# Rapid Prototyping and Risk

In order to minimize risk, a quick prototype of the system is made. A quick mockup of the system, shown below, is produced to ensure that the project is headed in the right direction and reduce overall risk in the early development phase of the project.

The prototype has also helped us view how the final project will look like and has inspired us to come up with ideas on what a user should expect from a schedule planner in terms of features, layout and ease-of-use.

User Interface Mockup

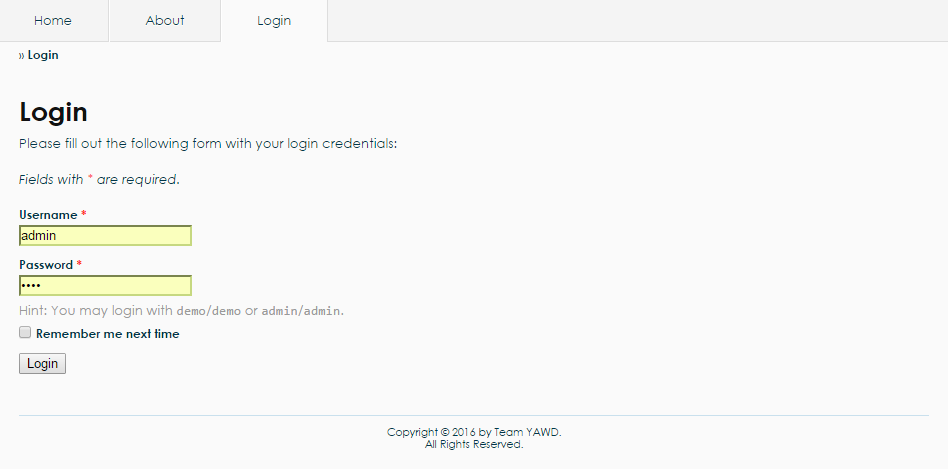
Main Layout

****

*Figure 1. Main layout of the system*

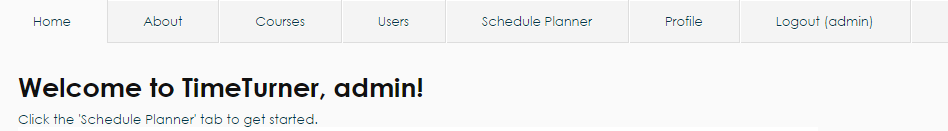
Upon accessing the TimeTurner website, the user is welcomed with a message and has the choice to login via the tab or by clicking the “click here” link. These will redirect the user to the Login page. The Yii framework was used to generate this layout, as it already contains a few themes.

Login Interface Mockup

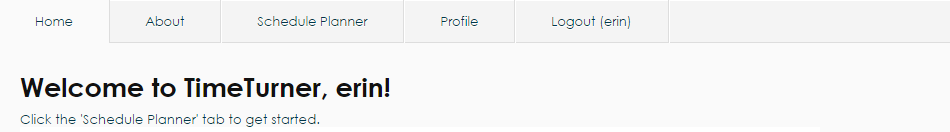


*Figure 2. Login interface mockup*

For the Login interface, there is an authentication form which has been generated by the Yii framework. The user may enter as an admin or as a student. A username and a password are required. Upon clicking the “Login” button, the system will query the database to check if the username and password are valid. The system will redirect to another page if the login is successful, else it will display an error message.



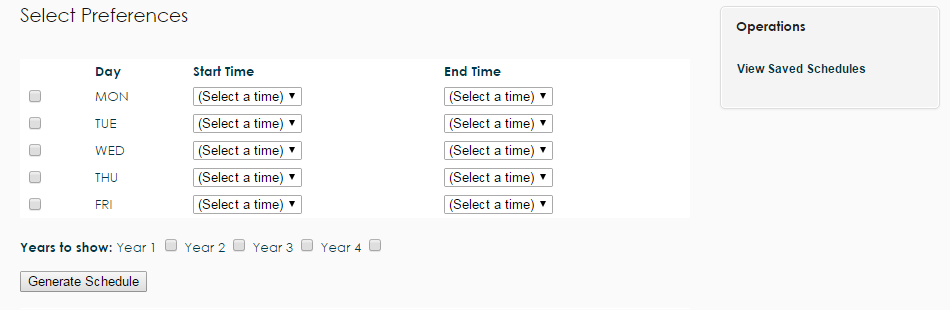
*Figure 3. User menu for an administrator*



*Figure 4. User menu for a student*

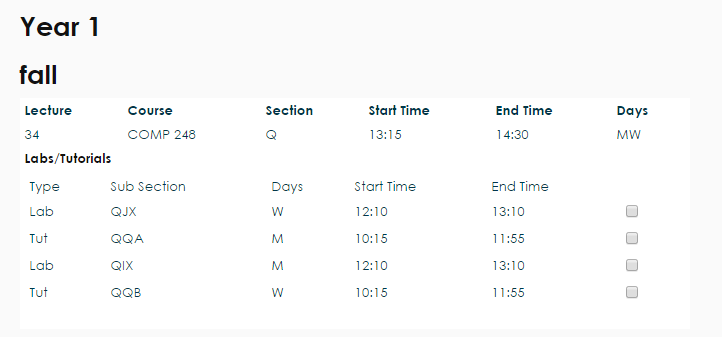
If the administrator logs in, additional tabs appear on top. These are “Courses”, “Users”, “Schedule Planner”, “Profile” as well as a “Logout” tab, as shown in Figure 3. If logged in as a student, the user cannot see the “Courses” and “Users” tabs, as shown in Figure 4, as these can only be modified by the administrator.

Generate Schedule Interface Mockup



*Figure 5. Generate schedule interface mockup*

All users have a “Schedule Planner” tab, which upon clicking, redirects the user to the page as shown in Figure 5. The user can select their preferences by checking one or more days of the week and select a starting and ending time. The user has the option of choosing which years can be displayed. Additionally, the “Viewed Saved Schedules” operation on the right navigation bar can be selected to view schedules that were previously generated and saved.



*Figure 6. Classes sections view*

Upon clicking the “Generate Schedule” button, classes are displayed along with their sections, as shown in Figure 6. The user may check the boxes next to the preferred sections.

Risk

Framework

One of the main risks involved in this project is the use of an unfamiliar framework. The current framework in use is Yii, which is based on MVC architecture in PHP. Due to the fact that most of the development team has never worked with a large framework, the learning curve tends to be very high. As such, possible delays might arise during development due to unforeseen problems that might come up while working with Yii. Though one of the major advantages of using this framework, is that it is loaded with features that make development much quicker in the long run; entire modules can be created on the fly with little effort. Depending on the learning curve progress, Yii can either create a healthy environment for development or hurt production..

Time Constraint

As deadline approaches, the realization of some features of software will have to be scoped out; only certain core components will be placed in order to meet a good portion of the major requirements of the scheduler.

Control Version System

Since the control version system in use is GitHub, the development team opted to use PhpStorm 10, an IDE provided by JetBrains, since it is integrated with Git. This adds up to the learning curve, since the team members have to learn an unfamiliar program, as well as how it works with Git. Using a framework requires working with many files, thus the project needs to be carefully managed. At times, PhpStorm 10 would not synchronize properly with the project’s repository on GitHub, which adds up to the time required to spend on learning and working on other components of the project. Therefore, learning how to properly use Git with PhpStorm can arise in possible delays during development due to team members’ inexperience with them.

Server Uptime

During team meetings, the production environment occurs at school. Often times, the network does not work properly as many students and employees try to access Internet at the same time. This would cause a decrease in productive time. Team members also have their own production environment at home. However, if a power outage occurs, the server computer may crash; hindering deployment and causing further delays.